

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31482 A2

(51) International Patent Classification⁷: **G06F 17/00**

(21) International Application Number: **PCT/US00/29146**

(22) International Filing Date: 20 October 2000 (20.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/425,579 22 October 1999 (22.10.1999) US

(71) Applicant: **GENERAL ELECTRIC CAPITAL CORPORATION** [US/US]; 44 Old Ridgebury Road, Danbury, CT 06810-5105 (US).

(72) Inventors: **ARDITTI, Mark, Lee**; 115 Walnut Grove Road, Ridgefield, CT 06877 (US). **WILSON, Judith, A.**; 20 Plumwood Road, Briarcliff Manor, NY 10510 (US). **FAIELLA, Steven**; 55 Mill Plain Road, Unit 22-4, Danbury, CT 06811 (US). **PRICE, David, K.**; 151 Lazy Brook Road, Monroe, CT 06468 (US).

(74) Agents: **CHASKIN, Jay, L. et al.**; General Electric Company, 3135 Easton Turnpike W3C, Fairfield, CT 06431 (US).

(81) Designated States (*national*): AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW.

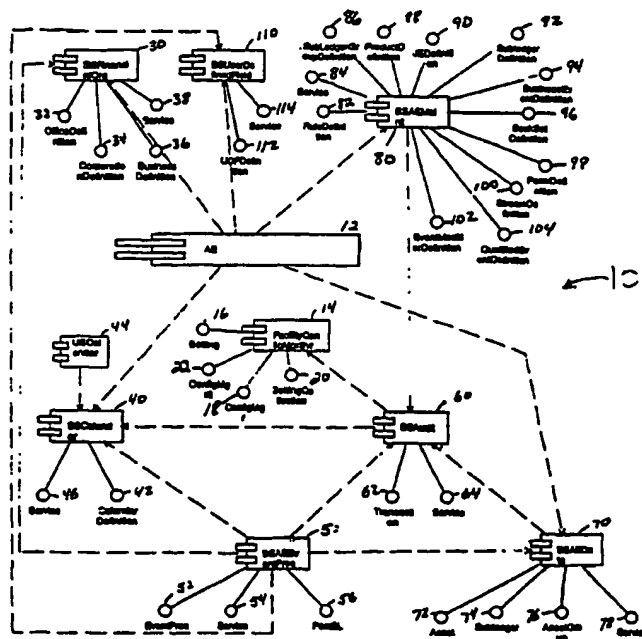
(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— Without international search report and to be republished upon receipt of that report.

[Continued on next page]

(54) Title: **LEASE AND LOAN SUB-LEDGER ACCOUNTING METHODS AND SYSTEM**



(57) Abstract: A lease and loan sub-ledger accounting system (10) that provides sub-ledger transaction detail for asset level accounting is described. The accounting system includes a lease and loan accounting engine (12), a plurality of component object model (COMTM) enabled sub-ledger accounting components, and a plurality of programmatic interfaces (140) enabling communication between the accounting components and the accounting engine.

package 30 further includes a business definition class 36 containing methods defining a business to accounting engine 12. Financial organization package 30 also includes a service class 38, which provides service to financial organization package 30 by retrieving data organization data from accounting engine 12.

5 Financial organization package 30 and classes described above as well as package and class definitions that follow are described in further technical detail in Appendix A titled Accounting Engine Package Documentation. The descriptions set forth in Appendix B are descriptions of the various accounting functions contained within accounting engine 12.

10 Lease and loan sub-ledger accounting system 10 also includes a calendar package 40 to provide support for multiple fiscal calendars. Calendar package 40 includes a calendar definition class 42 used to identify a fiscal closing date for a bookset and resolve key activity dates used for periodic processes such as bank holidays. If an asset uses multiple booksets, those booksets all use the same
15 calendar. A user service calendar package 44 is also included in calendar package 40 to allow calendar package 40 to run without a complete install of accounting engine 12. A service class 46 which provides service to calendar package 40 is also included.

20 Lease and loan sub-ledger accounting system 10 also includes an event processor package 50 to recognize a financial asset such as a piece of equipment, a lease, or a loan to also support account level or asset level accounting. Event processor package 50 includes an event processor class 52 containing methods used to interface with accounting engine 12 that require creation of journal entries and that are fundamental to transaction processing between the operational system and
25 accounting engine 12. A service class 54 is included in event processor package 50 that contains encapsulated retrieval methods for event processor 50. Event processor 50 further contains a post sub-ledger class 56, which is a controller class used to create or modify sub-ledgers and their supporting transaction detail.

is a sub-ledger definition class 92 containing methods for creating, updating, and using a sub-ledger chart of accounts in accounting engine 12. Maintenance package 80 further includes a business event definition class 94 that contains methods for creating, updating, and using a business event in accounting engine 12. A book set
5 definition class 96 in maintenance package 80 contains methods for creating, updating, and using a book set in accounting engine 12 which enables accounting system 10 to use multiple types of generally accepted accounting principles.

Maintenance package 80 still further includes a parameter definition class 98 containing methods for creating, updating, and using a parameter in
10 accounting engine 12. A stream definition class 100 in maintenance package 80 contains methods for creating, updating, and using data streams to compress the high volume of information for supporting asset level accounting and reducing storage requirements in accounting engine 12. An event modifier definition class 102 in
15 maintenance package 80 contains methods for creating, updating, and getting an event modifier such as country, business, or product specific exceptions to an accounting event in accounting engine 12. A qualified event definition class 104 in maintenance package 80 is used to describe specific event combinations based on a financial product by creating product and business event association in accounting engine 12 using journal entries and event modifiers.

20 Qualified event definition class 104 of maintenance package 80 together with event processor package 50 provide a flexible event driven process model to allow accounting engine 12 to derive the correct accounting entry for a lease or loan accounting event.

In addition, maintenance package 80 and event processor package 50
25 provide user defined finance rules for determining a correct type of accounting entry based on existing information and calculation rules to support financial calculations needed to properly account for leases and loans in multiple business organizations and countries.

Other examples of form interfaces are: Sub-ledger Chart Groups 170, used to add, update, delete, and display sub-ledger groups. Qualified Event Inquiry 172, Journal Entry Maintenance 174, used to maintain journal entry headers, Event Modifier Maintenance 176, Organization Maintenance 178, and Sub-ledger Chart of Accounts 180 used to add, update, and delete subledger chart of accounts. Organization Maintenance 178 form interface allows access to other form interfaces such as Office Maintenance 182 and Business Maintenance 184. Office Maintenance 182 form interface further allows access to Office Maintenance Part Two 186 and Business Maintenance 184 form interface allows access to Business Add 188 form interface.

Other form interfaces shown in Figure 2 are Rule Maintenance 190 and Qualified Event Maintenance 192. Rule Maintenance 190 form interface allows access to form interface Rule Maintenance Lines 194 which in turn allows access to form interface Parameter Maintenance 196. Qualified Event Maintenance 192 form interface allows access to other form interfaces such as Qualified Event Lines 198 and Product Pick 200. Qualified Event Lines 198 allows access to form interface Qualified Event Parameters Maintenance 202. Qualified Event Parameters Maintenance 202 also allows access to form interface Parameter Maintenance 196.

Lease and loan sub-ledger accounting system 10 is capable of supporting multiple pricing models and multiple operational systems. That capability provides stability when used with the accounting system of choice by isolating accounting engine 12 from the operational system. Therefore, the ability to change operational systems without negatively impacting the accounting system is enhanced. In addition, asset level detail is provided that is required for complex lease and loan transactions.

While the invention has been described in terms of various specific embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit scope of the claims.

IAset Class**Description**

The Asset can represent a physical piece of equipment or a financial entity such as a loan or an unapplied cash account. All Assets will have a corresponding Asset represented on the source (ATLAS) system.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

Long Create(ADOR.Recordset byval arsAsset, Long alTransid)

Class: IAsset

Description:

This will create one asset using the recordset.

This operation will be invoked after IService.GetAsset (0) has been used to return an empty recordset which can be populated with valid asset data by the operational (ATLAS) system.

This will return the asset entity id for the asset created.

Inputs: byval arsAsset -
alTransid -

Outputs: None

Returns: Long

Long Update(ADOR.Recordset byval arsAsset, Long alTransid)

Class: IAsset

Description:

Modify an asset using the ADOR.Recordset.

Inputs: byval arsAsset -
alTransid -

Outputs: None

Returns: Long

String Ping()

Class: IAsset

Description:

	byval astrExtAssetGroupRef -
	byval avAssetEntityIdat -
	byref TransId -
<u>Outputs:</u>	None
<u>Returns:</u>	long

Boolean Delete(long alEntityId)

<u>Class:</u>	IAssetGroup
<u>Description:</u>	This will remove an Asset Group (not the individual assets) from the AE.
<u>Inputs:</u>	alEntityId -
<u>Outputs:</u>	None
<u>Returns:</u>	Boolean

short RemoveAssets(Long alEntityId, VariantArray avAssetEntityIDs)

<u>Class:</u>	IAssetGroup
<u>Description:</u>	Remove one or more Assets from an Asset Group using the list of assets specified in the array. If 'ALL' is specified then all Assets will be disassociated with this Asset Group. Return a count of the assets removed from the Asset Group.
<u>Inputs:</u>	alEntityId - avAssetEntityIDs -
<u>Outputs:</u>	None
<u>Returns:</u>	short

String Ping()

<u>Class:</u>	IAssetGroup
<u>Description:</u>	Return a string indicating whether this object is instantiated.
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	String

IService Class

Description

This is the service component of the Accounting Engine. This will service: Assets, Subledgers, The Event Processor, and streams.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

String Ping()

<u>Class:</u>	IService
<u>Description:</u>	Return a string indicating whether this object is instantiated.
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	String

ADOR.Recordset GetAllAssetBooksetsByID(long byval alEntityID)

<u>Class:</u>	IService
<u>Description:</u>	Get all of the Booksets associated with an asset by the asset id. Each asset can be used to make entries in multiple booksets.
<u>Inputs:</u>	byval alEntityID -
<u>Outputs:</u>	None
<u>Returns:</u>	ADOR.Recordset

ADOR.Recordset GetAllAssetGroupTypes()

<u>Class:</u>	IService
<u>Description:</u>	Get all of the Asset Group Types in the AE database. Group types are used to identify / stratify the asset groups that have been created. e.g. ATLAS may create a loan Asset Group and a Customer Asset Group. Each of these may have the EntityId 1234567 in ATLAS, since they represent different data. The Ae needs to know what kind of group type (Customer or Loan) to retrieve if asset group value 1234567 is specified.
<u>Inputs:</u>	None
<u>Outputs:</u>	None

Returns: ADOR.Recordset

ADOR.Recordset GetAllAssetsByEntityID(long byval alEntityID)

Class: IService

Description:

Get a list of all of the products that are associated with this asset. An asset may behave like a tax product in one set of books and a loan product in another set of books.

Inputs: byval alEntityID -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllAssetTypes()

Class: IService

Description:

This is used to return all of the asset types in the AE. This is used to subclass assets. Is this asset a loan, a piece of equipment or a suspense account?

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllOfficeCorps()

Class: IService

Description:

Get all of the Office Corps in the AE. This is the junction of valid office / corp combinations.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllSLBalancesByAssetId(long byval alAssetEntityId, byval astrYear as string, BSAEDataSvcPeriodEnum byval aPeriod)

Class: IService

Description:

This will return a series of Subledger balances for a single asset and a single period. This needs to include the Subledger name, EntityID and amount for every SL found for the asset.

Inputs: byval alAssetEntityId -
byval astrYear as string -
byval aPeriod -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetSLDetailByAssetGroupID(long byval alAssetGroupID, long byval alCOAEntityID, date byval adteFrom, date byval adteTo)

Class: IService

Description:

This will return all of the Subledger details found for a subledger for the specified asset and date range. Include rows matching the from and to date in the result set. Include subledger header information

Inputs:

byval alAssetGroupID -
byval alCOAEntityID -
byval adteFrom -
byval adteTo -

Outputs:

None

Returns:

ADOR.Recordset

ADOR.Recordset GetSLDetailBySLandAsset(long byval AssetEntityID, long byval alCOAEntityID, date byval adteFrom, date byval adteTo)

Class: IService

Description:

This will return the Subledger details found for a single subledger for the specified asset and date range. Include rows matching the from and to date in the result set. Include the subledger header information.

Inputs:

byval AssetEntityID -
byval alCOAEntityID -
byval adteFrom -
byval adteTo -

Outputs:

None

Returns:

ADOR.Recordset

ADOR.Recordset GetSLDetailBySLGroupAsset(long byval alAssetID, long byval alSLGroupID, date byval adteFrom, date byval adteTo)

Class: IService

Description:

Get the Subledger detail for a subledger group associated with a single asset.

Inputs:

byval alAssetID -
byval alSLGroupID -
byval adteFrom -
byval adteTo -

Outputs:

None

Returns:

ADOR.Recordset

This will return all of the balances found on a single subledger account for the year and asset passed in to this method.

Inputs:

byval alAssetId -
byval alCOAEntityID -
byval astrYear -

Outputs:

None

Returns:

ADOR.Recordset

Create or update the Subledger balance and create a corresponding Subledger detail. This is an all or nothing unit of work.

The variant array contains all of the data needed to post one or more debit / credit pairs. It will always work on at least one debit and one credit.

Rules:

1. If the SL_Balance does not exist then invoke create to create the SL for this asset and then invoke CreateYear to create a new year of SL_balance for this asset.
2. Fiscal period needs to be resolved using the effective date. All posting will occur in the current fiscal period for this calendar.
3. Post the balance to the fiscal month. The S/L balance needs to be propagated forward from the transaction date for all months in the transaction year.
4. Invoke CreateDetail to create the SL_Detail row.
5. abooReverseOperator needs to be inspected to determine how acurAmount should be signed. If abooReverseOperator = true then acurAmount should be reversed by multiplying by -1.
6. For credits subtract the amount being posted, for debits add the amount. Since Post is calculating the correct operator, pass the correct signed amount to CreateDetail
7. Return the alTransId created by the Audit component as long.

Inputs:

byval alEntityId -
byval avarPostData -

Outputs:

None

Returns:

Long

PrivateAccess Methods

Long CreateDetail(long byval alSLBalanceEntityId, date byval adteEffective, currency byval acurAmount, string byval astrDebitCredit, long byval alBankEntityId, long byval alJEEntityId, date byval adtePeriod)

Class:

ISubledger

Description:

This is the only method used to create the supporting detail for the sl balance. This is an important Audit point.

This will be invoked by Post, Rollover.

All fields are required except for Bank.

Return the EntityID of the debit or credit created.

5. Return aiEntityId as long.

Inputs:

byval aiSLBalanceEntityId -

Outputs:

byval adteYear -

Returns:

None

long

BSAEMaint Packag Overview**D scription**

This package contains the business service classes required to support the user maintenance of Accounting Engine data.

Classes

- IBooksetDefinition
- IBusinessEventDefinition
- IEventModifierDefinition
- IJEDefinition
- IParmDefinition
- IProductDefinition
- IQualifiedEventDefinition
- IRuleDefinition
- IService
- IStreamDefinition
- ISubLedgerGroupDefinition
- ISubledgerDefintion

Subpackages

None

- Delete from Bookset where SQ_BOOKSET_ID = alEntityId.
- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

Update(Long byval alEntityId, string byval astrBooksetName, string byval astrBooksetDesc, integer byval alActiveId, Long byval alTaxTypeid, long byval alReportTypeid, Long byref alTransNbr, string byval astrDescription)

Class: IBooksetDefinition
Description:

This will update one Bookset in the Accounting Engine.

- Get today's date for adtsStatusDate
- Update Bookset

Inputs: byval alEntityId -
byval astrBooksetName -
byval astrBooksetDesc -
byval alActiveId -
byval alTaxTypeid -
byval alReportTypeid -
byref alTransNbr -
byval astrDescription -
Outputs: None
Returns: None

String Ping()

Class: IBooksetDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

Outputs: byref aITransNbr -
Returns: N n
 N ne

Update(Long byval aiEntityId, string byval astrEntityDesc, Long byref aITransNbr)

Class: IBusinessEventDefinition

Description:

This will update one Business Event in the Accounting Engine.

· Update Business_Event using astrEntityDesc

· aITransNbr = Call LogTransId.

Inputs: byval aiEntityId -
 byval astrEntityDesc -
 byref aITransNbr -

Outputs: None

Returns: None

String Ping()

Class: IBusinessEventDefinition

Description:

Return a string indicating whether this object is instantiated.

Inputs: None

Outputs: None

Returns: String

This will delete an Event Modifier, its associated Event Modifier Lines and the Event Modifier Parm List from the Accounting Engine.

Referential integrity will need to be enforced for the Qualified Event. It is only possible to delete an Event Modifier if there are no QE's that use it.

Inputs: byval aiEntityId -
byval arsLines -
byref aiTransNbr -
optional arsParms -
Outputs: None
Returns: None

String Ping()

Class: IEventModifierDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

Long Update(long byval aiEntityId, string byval astrName, string byval astrDesc, ADOR.Recordset byval arsLines, long byref aiTransId)

Class: IEventModifierDefinition
Description: Update the name, description or Event Modifier Lines for this Event Modifier.
Inputs: byval aiEntityId -
byval astrName -
byval astrDesc -
byval arsLines -
byref aiTransId -
Outputs: None
Returns: Long

PrivateAccess Methods

UpdateEventModData(ADOR.Recorset byval arsLines)

Class: IEventModifierDefinition
Description: Update the Event Modifier Lines or Parmas using a Recordset.
Inputs: byval arsLines -
Outputs: None
Returns: None

IJEDefinition Class**Description**

This interface contains the methods required to create or update a JE in the Accounting Engine. This will maintain the Journal Entry (Lookup table) entity: JE Name, description, (DR/CR pairs).

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long Create(string byval astrName, string byval astrDesc, boolean byval aboolManual, long byval alJENumber, ADOR.Recordset byval arsJEDetail, long byref alTransNbr)

Class:

IJEDefinition

Description:

This will define a Journal Entry to the Accounting Engine.

The ADOR.Recordset contains the list of debit/credit pairs for this JE.

1. Insert JE
2. Insert debit / credit pairs using ADOR.Recordset and the private method UpdateJEDetailRS.
3. alTransNbr = Call LogTransId.
4. Return ID as long

Inputs:

byval astrName -
byval astrDesc -
byval aboolManual -
byval alJENumber -
byval arsJEDetail -
byref alTransNbr -

Outputs:

None

Returns:

long

Delete(string byval alEntityId, long byref alTransNbr)

Class:

IJEDefinition

Description:

This will be used to add or remove debit / credit pairs from this JE.

It will always be necessary to get the ADOR.Recordset before using this method. Note: an empty recordset will be returned if there are no debit / credit pairs for this JE. This empty recordset can then be used in this method to insert debits and credits just as if this is an ordinary recordset update.

Inputs:

byval arsJEDetail -
byval aobjContext -
byval aobjDataClass -
byval aobjAudit -

Outputs:

None

Returns:

None

IParmDefinition Class

Description

This interface contains the methods required to create, update, and use a Parameter in the Accounting Engine. This will maintain the Parameter entities: Parameter Name, Description and Parameter Type.

Parameter Type needs to be initially populated using SQL. This will not change often enough to write the definition methods to support this table. Parameter Type will contain values like: String, Numeric, Currency.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrParmName, string byval astrParmDesc, string byval alEntityId, long byref alTransNbr, String byval astrParmTypeID)

Class:

IParmDefinition

Description:

This will create a Parameter in the Accounting Engine. Before a Product Business Event can refer to a Parameter it will be necessary to define (create) the Parm entity.

- Insert into PARM
- alTransNbr = Call LogTransId.
- Return ID as long

Inputs:

byval astrParmName -
byval astrParmDesc -
byval alEntityId -
byref alTransNbr -
byval astrParmTypeID -

Outputs:

None

Returns:

long

Delete(long byval alEntityId, long byref alTransNbr)

Class:

IParmDefinition

Description:

This will delete an Parm in the Accounting Engine.

- Delete from PARM
- alTransNbr = Call LogTransId.

IProductDefinition Class

Description

This interface contains the methods required to create, update, and use a Product in the Accounting Engine. This will maintain the Product (Lookup table) entity: Product name, description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(String byval astrName, string byval astrDesc, long byref aiTransNbr)

Class: IProductDefinition

Description: This will create a Product in the Accounting Engine. Before any Asset can refer to a Product, it will be necessary to define (create) the Product entity.

- Validate required fields: astrName and astrDesc
- Check for duplicate on Product_Name
- Insert into Product_AE
- aiTransNbr = Call LogTransId.
- Return ID as long

Inputs: byval astrName -
byval astrDesc -
byref aiTransNbr -

Outputs: None

Returns: long

Delete(Long byval aiEntityId, long byref aiTransNbr)

Class: IProductDefinition

Description: This will delete a Product in the Accounting Engine. Referential integrity needs to be enforced.

- Delete from Product_AE
- aiTransNbr = Call LogTransId.

Inputs: byval aiEntityId -
byref aiTransNbr -

Outputs: None

Returns: None

IQualifiedEventDefinition Class

Description

This is where the pieces come together: product, business event, je, event modifier and Rules. There is no Update method for this interface. It will be necessary to Delete and Create a new Qualified Event.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

CreateProductBusinessEvent(long byval alBusinessEventEntityId, long byval alProductEntityId, long byref alTransNbr, ADOR.Recordset byval optional arsParms)

Class:

IQualifiedEventDefinition

Description:

This will create a Product and Business Event association in the Accounting Engine. Before any Qualified Event can refer to a Business Event, it will be necessary to define (create) the Business Event entity, the Product Entity and associate the Business Event and Product.

Return the Entity Id for the Business Event Product that has been created.

Insert Business Event / Product
Insert Parms using ADOR.Recordset
alTransNbr = Call LogTransId.

Inputs:

byval alBusinessEventEntityId -
byval alProductEntityId -
byref alTransNbr -
byval optional arsParms -

Outputs:

None

Returns:

None

DeleteProductBusinessEvent(long byval alBusinessEventEntityId, long byval alProductEntityId, long byref alTransNbr)

Class:

IQualifiedEventDefinition

Description:

This will delete a Product and Business Event association from the Accounting Engine and the

byval alProductId -
 byval alEventModId -
 alRuleId -
 byval alJEID -
 byval alJENonEarnId -
 aRSBooksets -
 byval alTransId -
 optional byval aRSRuleVars -
Outputs: None
Returns: None

UpdateQualEventLine(long byval alQualEventId, long byval alBusinessEventId,
 long byval alProductId, long byval alEventModId, long byval alRuleId, long byval
 alJEID, long byval alJENonEarnId, string byval astrEntryName, ADOR.Recordset
 byval arsBooksets, long byref alTransId, ADOR.Recordset optional byval
 arsRuleVars)

Class: IQualifiedEventDefinition
Description: Update a Qualified Event Line.

Inputs: byval alQualEventId -
 byval alBusinessEventId -
 byval alProductId -
 byval alEventModId -
 byval alRuleId -
 byval alJEID -
 byval alJENonEarnId -
 byval astrEntryName -
 byval arsBooksets -
 byref alTransId -
 optional byval arsRuleVars -
Outputs: None
Returns: None

DeleteQualEventLine(long byval alQualEventId, long byref alTransId)

Class: IQualifiedEventDefinition
Description:

Delete a specific Qualified Event line.
Inputs: byval alQualEventId -
 byref alTransId -
Outputs: None
Returns: None

long CreateRuleVar(long byval alVarTypeId, long byval alQualEventId, long byval
 alRuleLineId, long byval alVarSeqNum, long byval alPBEParmId, long byval
 alDBFieldId, string byval strConstantValue, long alOrigRuleLine, long byref
 alTransId)

Class: IQualifiedEventDefinition

IRuleDefinition Class**Description**

This interface contains the methods required to create, update, and use a Rule in the Accounting Engine. This will maintain the Rule (Lookup table) entity: Rule name, description and the Rule lines that define the Rule.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long CreateHdr(string byval astrRuleName, string byval astrRuleDesc, long byref aiTransid)

Class:**IRuleDefinition****Description:**

This will create a Rule in the Accounting Engine. Before an Event can refer to a Rule, it will be necessary to define (create) the Rule entity.

Rule is a Rule header and lines.

Return the Entity Id for the Rule created, not for the Rule lines.

Inputs:**byval astrRuleName -****byval astrRuleDesc -****byref aiTransid -****Outputs:****None****Returns:****long**

Long AddRuleLine(Long byval aiRuleId, Long byval aiVerbLUID, String byval astrRuleDest, Long byval aiLineSeq, ADOR.Recordset byval aRSRuleVars, Long byref aiTransid)

Class:**IRuleDefinition****Description:**

Add a single Rule line for a Rule. The Rule line is used to define the Rule Verb, Destination and the variables that need to be resolved to process the Rule.

Inputs:**byval aiRuleId -****byval aiVerbLUID -****byval astrRuleDest -****byval aiLineSeq -**

Outputs: byval aRSRul Vars -
Returns: byref aTransID -
 None
 Long

DeleteRule(long byval aEntityId, long byref aTransId)

Class: IRuleDefinition
Description:

This will delete a Rule and all of its Rule lines in the Accounting Engine.
 This is all or nothing behaviour. Rule lines can not be deleted if the Rule delete fails for any reason (including enforced referential integrity).

Inputs: byval aEntityId -
 byref aTransId -
Outputs: None
Returns: None

String Ping()

Class: IRuleDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

UpdateHdr(long byval aEntityId, string byval astrName, string byval astrRuleDesc, long byref aTransNbr)

Class: IRuleDefinition
Description:

This will update one Rule Name or description in the Accounting Engine.

Inputs: byval aEntityId -
 byval astrName -
 byval astrRuleDesc -
 byref aTransNbr -
Outputs: None
Returns: None

DeleteLine(long byval aEntityId, byref aTransId)

Class: IRuleDefinition
Description:

IService Class**Description**

This interface contains the methods required to "service" BSAEMaint. We need to review services against windows to make sure we can populate all fields that we have on existing windows.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

Long DoesSubledgerCodeExist(string byval astrSubledgerCodeExist)

Class: IService

Description:

This will check for the existance of a subledger code and return the entityID for the subledger code if it is found.

Inputs: byval astrSubledgerCodeExist -

Outputs: None

Returns: Long

ADOR.Recordset GetAllAccountingPeriods()

Class: IService

Description:

Get all of the valid Accounting Periods and Dates that may be used in a Rule. e.g. CurrYear, PriorYear, Today, CurrMonth, etc.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllBooksets()

Class: IService

Description:

This will get all Booksets in the AE.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllBusinessEvents()

ADOR.Recordset GetAllRules()

Class: IService
Description: This will get a list of all Rules in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllSubledgers()

Class: IService
Description: This will get a list of all Subledgers in the Chart of Accounts in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetBooksetById(long byval alEntityId)

Class: IService
Description: This will get a Bookset in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetBusinessEventById(long byval alEntityId)

Class: IService
Description: This will get a Business Event in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetEventModifierById(long byval alEntityId)

Class: IService
Description: This will get an Event Modifier using the Entity Id of the Event Modifier.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetEventModifierLinesById(Long byval alEntityId)

Class: IService
Description: This will return the Event modifier lines in an ADOR.Recordset.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetRuleVarsByRuleLineId(long byval aiEntityId)

Class: IService

Description: Get the Rule line variables associated with a single rule line.

Inputs: byval aiEntityId -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetStreamTypeById(long byval aiEntityID)

Class: IService

Description: Get the stream type by the stream type id.

Inputs: byval aiEntityID -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetSubledgerById(long byval aiEntityId)

Class: IService

Description: This will get a Subledger from the Chart of Accounts in the AE.

Inputs: byval aiEntityId -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetSubledgerWithFilter(string byval astrColumn, string byval astrMatchPattern)

Class: IService

Description: Return SI by SLcode using a SQL 'Like' Subledger code.

Inputs: byval astrColumn -

byval astrMatchPattern -

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetSublForGroup(long byval aiEntityId)

Class: IService

Description: This is return all of the subledgers in the Chart of Accounts for a single Subledger Group.

Inputs: byval aiEntityId -

Outputs: byval alBusinessEventID -
Returns: None
ADOR.Recordset

ADOR.Recordset GetQERuleVarsByQELineId(long byval alEntityId)

Class: IService

Description:

QE Qualified Event

Get all of the rule variables associated with a qualified event line using the QE Id.

Inputs: byval alEntityId -

Outputs: None

Returns: ADOR.Recordset

Boolean GetPBE(string byval astrProduct, string byval astrBusinessEvent, long byref alPBEID)

Class: IService

Description:

Get a single product business event by specifying the product and business event.

Inputs: This is not completed yet.

byval astrProduct -

byval astrBusinessEvent -

byref alPBEID -

Outputs: None

Returns: Boolean

**Update(long byval aIEntityId, String byval astrStreamDesc, long byval aITransID,
String byval astrStreamName)**

Class: IStreamDefinition

Description:

This will update the name or description for one Stream
in the Accounting Engine.

Inputs:

byval aIEntityId -
byval astrStreamDesc -
byval aITransID -
byval astrStreamName -

Outputs:

None

Returns:

None

String Ping()

Class:

IStreamDefinition

Description:

Return a string indicating whether this object is
instantiated.

Inputs:

None

Outputs:

None

Returns:

String

Inputs: byval astrName -
byval astrDesc -
byref alTransNbr -
Outputs: None
Returns: long

Delete(long byval alEntityId, long byref alTransNbr)

Class: ISubLedgerGroupDefinition

Description:

Delete a Subledger Group from the Accounting Engine.

· Delete from SL_Group

· alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -

Outputs: None

Returns: None

RemoveSubledger(long byval alSubledgerGroupId, long byval alSubledgerId, long byref alTransNbr)

Class: ISubLedgerGroupDefinition

Description:

Remove a subledger from this Subledger Group.

· Delete from SL_Group_Subledgers where
SQ_CHART_OF_ACCOUNT_ID = alSubledgerId and
SQ_SL_GROUP_ID = alSLGroupId.

· alTransNbr = Call LogTransId.

Inputs: byval alSubledgerGroupId -
byval alSubledgerId -
byref alTransNbr -

Outputs: None

Returns: None

Update(Long byval alEntityId, String byval astrDesc, Long byref alTransNbr)

Class: ISubLedgerGroupDefinition

Description:

Update the description for a subledger group.

· update SL_GROUP

· alTransNbr = Call LogTransID

Inputs: The name for a sl group can not be changed???
byval alEntityId -
byval astrDesc -
byref alTransNbr -

ISubledgerDefintion Class**Description**

This interface contains the methods required to create, update, and use the Subledger Chart of Accounts in the Accounting Engine. This will maintain the Subledger (Lookup table) entity: Chart of Accounts, name, description.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long Create(string byval astrRollupId, string byval astrTypeMemoGI, string abyval astrTypeALER, string byval astrSLCode, string byval astrActiveId, string byval astrSubLedgerName, long byval aITransferId, string byval astrCrossref, long byref aITransNbr)

Class:

ISubledgerDefintion

Description:

This will create a Subledger in the Accounting Engine Chart of Accounts. Before any Asset can be refer to a Subledger, it will be necessary to define (create) the Subledger entity. This will return the entity Id as a long.

- Test for required fields (except for astrCrossRef).
- Insert into SL_Chart_of_Accounts values.
- aITransNbr = Call LogTransId.
- Return ID as long

Inputs:

byval astrRollupId -
byval astrTypeMemoGI -
abyval astrTypeALER -
byval astrSLCode -
byval astrActiveId -
byval astrSubLedgerName -
byval aITransferId -
byval astrCrossref -
byref aITransNbr -

Outputs:

None

Returns:

long

Delete(long byval aIEntityId, long byref aITransNbr)

Outputs: None
Returns: String

UpdateRS(ador.recordset byval arsSubledger)

Class: ISubledgerDefintlon

Description:

Use ADOR recordset to add, uipdateo r delete a record from the database.

Inputs: byval arsSubledger -

Outputs: None

Returns: None

IServiceClass**Description**

IService is used to retrieve data.

SQ_TRANSACTION_NBR is the database field used to identify the AE Transaction. There can be multiple rows in the Transaction table for each AE transaction.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

ADOR.Recordset GetTransByDate(date byval adteFrom, Date byval adteTo)

Class:

IService

Description:

This will get all transactions in the Audit Component for a given date range. This will return the transaction details as an ADOR.Recordset.

Get Timestamp_Date = > adteFrom and =< adteTo
Order by TIMESTAMP_DATE descending

Inputs:

byval adteFrom -
byval adteTo -

Outputs:

None

Returns:

ADOR.Recordset

ADOR.Recordset GetTransByEntityId(long byval alEntityId)

Class:

IService

Description:

This will get a transaction entity in the Audit Component. This will return the details of a single row in the Transaction table as an ADOR.Recordset.

alEntityId The Entity Id of the specific row in the transaction table being returned.

Inputs:

byval alEntityId -

Outputs:

None

Returns:

ADOR.Recordset

ADOR.Recordset GetTransByTransNbr(Long byval alTransNbr)

Class:

IService

Transaction Class

Description

This interface contains the methods required to create, and use the unique transaction id in the Audit Component. This component may be used by multiple components. This will be recorded on all Accounting Engine entities, when they are created or updated for a complete transaction audit trail. This will also be useful for enabling Undo functionality.

Public enum eAuditTransTypes

```
ecAdd
ecCreate
ecDelete
ecRemove
ecUpdate
end enum
```

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long LogTrans(Long byval alFacilityId, eAuditTransTypes byval aiTransType, string byval astrEntityName, long byval alEntityId, Long byref optional aiTransNbr = 0)

Class:

Description:

ITransaction

This will generate a unique transaction number with details about the transaction for audit purposes and return the Entity Id for the transaction generated in this function. The userid will be obtained from the MTS context Object. The date-timestamp will be obtained from the system. The aiTransNbr will be generated if it is zero.

aiTransType The type of database activity that was performed by this transaction as defined in the eAuditTransTypes enum.

astrEntityName The name of entity that is associated with the Entity ID stored for this transaction.

alEntityId The Entity ID for the entity involved in this activity.

- Get UserId from MTS context Object.
- Get Date-timestamp from system.
- If aiTransNbr = zero

tn = create new transaction number

Else

tn = aiTransNbr

BSCalendar Package Overview**Descripti n****Classes**

ICalendarDefinition
IService

Subpackages

None

avFiscalStartMonths This is an array of 11 dates which represent the fiscal start for each month, after the first month of the year. The first month of the year is derived from **aiFiscalYear** and the Fiscal Start month and day on the **FiscalCalendar**.

adteFiscalYearEnd This is the last day of the fiscal year being created.

- **Update_Date** = Today
- Update_Userid** = **objectcontext.OriginalCaller**
- Insert **Calendar_Date** records for each day in **aiFiscalYear**.
- Insert **CALENDAR_ACTIVITY_DATE** for each start date to create the junction between "FISCAL MONTH START" and the date
- return **id**

Inputs: **byval aiCalendarEntityId** -
byval avFiscalStartMonths -
byval byref aiTranNbr -

Outputs: None
Returns: None

Long CreateActivityType(string byval astrActivityTypeName, string astrActivityTypeDesc, Integer aiReservedInd)

Class: **ICalendarDefinition**

Description: This will create a Calendar Activity Type in the Calendar component.

Type

- **Update_Date** = Today
- Update_Userid** = **objectcontext.OriginalCaller**
- Insert into **Calendar_Activity_Type**
- return **id**

Inputs: **byval astrActivityTypeName** -
astrActivityTypeDesc -
aiReservedInd -

Outputs: None
Returns: Long

long AddDateActivity(long byval aiActivityTypeEntityId, long byval aiDateEntityId)

Class: **ICalendarDefinition**

Description: This is used to connect a Calendar Activity Type to a Calendar Date for a single Calendar.

The Entity Id for the Activity Type.

aiDateEntityId The Entity Id for the date that is being associated with an Activity Type.

Update_Date = Today

Calendar_Activity_Date rec rd from th Calendar component.

Inputs: byval aiDateEntityId -
byval aiActivityEntityId -
byval aiTransnbr -
Outputs: None
Returns: None

UpdateActivityType(long byval aiEntityId, string byval astrName, string byval astrDesc, Integer byval aiReservedInd)

Class: ICalendarDefinition
Description: This will update a Calendar ActivityType in the Calendar component.

Inputs: byval aiEntityId -
byval astrName -
byval astrDesc -
byval aiReservedInd -
Outputs: None
Returns: None

String Ping()

Class: ICalendarDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

UpdateCalendar(String byval astrCalendarName, string byval astrCalendarDesc, long byref aiTransnbr, Integer byval aiReservedInd)

Class: ICalendarDefinition
Description: This will update a Calendar in the Accounting Engine. Before any Asset or Corp can refer to a Calendar, it will be necessary to define (create) the Calendar entity.
Inputs: byval astrCalendarName -
byval astrCalendarDesc -
byref aiTransnbr -
byval aiReservedInd -
Outputs: None
Returns: None

Inputs: byval alEntityId -
astrFromDate -
asteT Date -
Outputs: None
Returns: ADOR.Recordset

String Ping()

Class: IService
Description: Return a string indicating whether this object is
instantiated.
Inputs: None
Outputs: None
Returns: String

**ADOR.Recordset GetAllActivityByDateRange(long byval alEntityId, date
astrFromDate, date asteToDate, long byval alActivityTypeId)**

Class: IService
Description: Return Calendar, and Activity Dates for one Activity
Type and a date range.
Inputs: byval alEntityId -
astrFromDate -
asteToDate -
byval alActivityTypeId -
Outputs: None
Returns: ADOR.Recordset

ICurrencyDefinition Class**Description**

This interface contains the methods required to create, update, and use a Currency in the Accounting Engine. This will maintain the Currency, Rounding Rules and Currency Rate (Lookup table) entities: Currency name, description, rates, rounding rules, description. It will be necessary to define a valid Rounding Rule before creating a Currency.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long CreateCurrency(String byval astrCurrencyName, string byval
alRoundingRuleEntityId, long byval alTransNbr)

Class: ICurrencyDefinition

Description: This will create a Currency in the Accounting Engine.
Before any Asset can be refer to a Currency, it will be
necessary to define (create) the currency and a
rounding rule.

Inputs: byval astrCurrencyName -
byval alRoundingRuleEntityId -
byval alTransNbr -

Outputs: None

Returns: long

CreateRate(Long byval alFromEntityId, long byval alToEntityId, date byval
adteEffectiveDate, string byval asConversionRate : single byval
astrConversionSrc, long byref alTransNbr)

Class: ICurrencyDefinition

Description: This will create a Currency Rate in the Accounting
Engine..Let's discuss. Do we have to get the From and
To Currency Id before we create a rate.

Inputs: byval alFromEntityId -
byval alToEntityId -
byval adteEffectiveDate -
byval astrConversionSrc -
byref alTransNbr -

Outputs: None

Returns: None

Returns: None

String Ping()

Class: ICurrencyDefinition

Description: Return a string indicating whether this object is instantiated.

Inputs: None

Outputs: None

Returns: String

DeleteCurrency(long byval alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition

Description: This will delete a Rounding Rule in the Accounting Engine. Referential integrity needs to be enforced.

Inputs: byval alEntityId -
byref alTransNbr -

Outputs: None

Returns: None

DeleteRoundingRule(long alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition

Description: This will delete a Rate Conversion from the Accounting Engine.

Inputs: alEntityId -
byref alTransNbr -

Outputs: None

Returns: None

DeleteRate(long byval alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition

Description: This will delete a Rate Conversion from the Accounting Engine.

Inputs: byval alEntityId -
byref alTransNbr -

Outputs: None

Returns: None

Long GetR undingRuleId(alEntityId)**Class:** IService**Description:** This will return the entity Id for a R unding Rule in th Accounting Engine.**Inputs:** alEntityId -**Outputs:** None**Returns:** Long**GetRate(long byval alentityId)****Class:** IService**Description:** This will get a Currency Rate in the Accounting Engine.**Inputs:** byval alentityId -**Outputs:** None**Returns:** None

IBusinessDefinition Class

Description

This interface contains the methods required to define a Business to the Accounting Engine.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrBusinessName, string byval astrBusinessDescription, long byref alTransid)

<u>Class:</u>	IBusinessDefinition
<u>Description:</u>	This will create a business in the Accounting Engine. Before any Office can be defined within a Business, it will be necessary to define (create) the Business entity. Insert the Business

<u>Inputs:</u>	byval astrBusinessName - byval astrBusinessDescription - byref alTransid -
<u>Outputs:</u>	None
<u>Returns:</u>	long

Delete(long byval alEntityId, long byref alTransid)

<u>Class:</u>	IBusinessDefinition
<u>Description:</u>	This will delete a Business from the Accounting Engine. Referential integrity needs to be enforced between this and the Corporation and the office. Delete the business

<u>Inputs:</u>	byval alEntityId - byref alTransid -
<u>Outputs:</u>	None
<u>Returns:</u>	None

Update(long byval alEntityId, string astrBusinessName, byval astrBusinessDescription, long byref alTransid)

<u>Class:</u>	IBusinessDefinition
----------------------	----------------------------

ICorporationDefinition Class**Description**

This interface contains the methods required to define a Corporation to the Accounting Engine. This will maintain the Corporation (Lookup table) entity.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

Create(string byval astrCorpName, string byval astrCorpDescription, long byval alCalendarId, long byref alTransId)

<u>Class:</u>	ICorporationDefinition
<u>Description:</u>	This will create a Corporation in the Accounting Engine. Before any Asset, Office, or Business can refer to a Corporation., it will be necessary to define (create) the Corporation entity. This will return the Entity Id as a long.
<u>Inputs:</u>	<p>alCalendarId The id of a fiscal calendar to be associated with this corporation.</p> <p>• IstrUserId = context.security.getoriginalcaller</p> <p>IstrDate = Date</p> <p>• Insert into Corp_Org</p> <p>byval astrCorpName -</p> <p>byval astrCorpDescription -</p> <p>byval alCalendarId -</p> <p>byref alTransId -</p>
<u>Outputs:</u>	None
<u>Returns:</u>	None

Delete(Long byval alEntityId, long byref alTransId)

<u>Class:</u>	ICorporationDefinition
<u>Description:</u>	This will Delete a Corporation defined to the Accounting Engine. Referential integrity needs to be enforced.
<u>Inputs:</u>	<p>byval alEntityId -</p> <p>byref alTransId -</p>
<u>Outputs:</u>	None

OfficeDefinition Class**Description**

This interface contains the methods required to define an Office to the Accounting Engine.
This will maintain the Office (Lookup table) entity.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

AddCorp(long byval alOfficeld, byval alCorpId, long byref alTransid)

Class: OfficeDefinition

Description:

This will add a junction relationship between an Office and a Corp.

· IstrUserId = context.security.getoriginalcaller

IstrDate = Date

· Insert into Office_Corp

Inputs: byval alOfficeld -

byval alCorpId -

byref alTransid -

Outputs: None

Returns: None

long Create(string byval astrOfficeName, string byval astrOfficeDescription, long byval alBusinessId, variant byval AVCorpIds)

Class: OfficeDefinition

Description:

This will create an Office in the Accounting Engine. Before any Asset can refer to an Office, it will be necessary to define (create) the Office entity. This will return the Entity Id as a long.

· IstrUserId = context.security.getoriginalcaller

IstrDate = Date

· Insert into Office

· Insert rows into Corp_Office for each corporation in the avCorpIds array.

· return Id

Inputs: byval astrOfficeName -
byval astrOfficeDescription -

IService Class**Description**

This provides the services for the Financial Organization component.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods****ADOR.Recordset GetAllBusinesses()**

Class: IService

Description: This will retrieve all businesses from the accounting engine returning them in a recordset.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllCorporations()

Class: IService

Description: This will retrieve all corporations from the accounting engine returning them in a recordset.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAllOffices()

Class: IService

Description: This will retrieve all offices from the accounting engine returning them in a recordset.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetBusinessByID(long byval aiEntityId)

Class: IService

BSImportExport Package Overview**Description****Classes**

IImportExport

Subpackages

None

Outputs: None
Returns: Long

Long ExportSLBalance(Long byval alBookSetId, String byval astrFileName, Date byval adtePeriod)

Class: IImportExport
Description: Export Sub ledger balances for an entire bookset, for a single period, to an external file. This will return the number of subledger balances written to astrFileName.

alBookSetId The bookset which will be exported.
adtePeriod The fiscal period used to filter this export.
astrFileName The output filename.

Inputs: byval alBookSetId -
 byval astrFileName -
 byval adtePeriod -

Outputs: None
Returns: Long

Long ExportSLDetailForDateRange(Long byval alBooksetID, Date byval adteFrom, Date byval adteTo, String byval astrFileName)

Class: IImportExport
Description: Export Sub ledger detail, by asset group, for an entire bookset, for a date range, to an external file. This will return the number of subledger details written to astrFileName.

Inputs: byval alBooksetID -
 byval adteFrom -
 byval adteTo -
 byval astrFileName -

Outputs: None
Returns: Long

Long ExportSLDetailForPeriod(Long byval alBooksetid, Date byval adtePeriod, String byval astrFileName)

Class: IImportExport
Description: Export Sub ledger detail, by Asset Group, for an entire bookset, for a period, to an external file. This will return the number of subledger details written to astrFileName.

Inputs: byval alBooksetid -
 byval adtePeriod -
 byval astrFileName -

Outputs: None

asset group does not exist, the asset group will be created and the assets will be added to it.

Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportAssetGroupUDFs(Long byval astrFileName)

Class: ImportExport
Description: Import Asset Group UDF's from astrFilename. This file will contain asset group name and UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

long ImportAssetUDFs(string byval astrFileName)

Class: ImportExport
Description: Import Asset UDF from astrFilename. This file will contain external asset reference and asset UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: long

Long ImportCurrencyRates(Long byval astrFileName)

Class: ImportExport
Description: Import currency conversion rates from astrFilename. This file will contain currency type, country, rates, and effective dates
Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportCurrencyRoundingRules(Long byval astrFileName)

Class: ImportExport
Description: Import currency conversion rounding rules from astrFilename. This file will contain currency type, country, and rounding rules.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

IService Class**Description**

This provides the services for the BSUDF component

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

ADOR.Recordset GetAllUDFNames()

Class: IService

Description:

This will get the list of all of the UDFs defined to the Accounting Engine.

Inputs: None

Outputs: None

Returns: ADOR.Recordset

String GetUDFValue(string byval astrName, long alEntityId, long alInstanceId)

Class: IService

Description:

This will get the value of a User Defined Field associated with an entity.

astrName The User defined field to be returned.
alEntityId The Id of the entity for which the UDF is being returned. (e.g. Id for "asset")
alInstanceId The Id of the specific entity instance for which the UDF is being returned. (e.g. Id for asset #123)

Inputs: byval astrName -

alEntityId -

alInstanceId -

Outputs: None

Returns: String

ADOR.RecordSet GetAllUDFTableName()

Class: IService

Description:

Inputs: byval astrNam -
Outputs: None
Returns: None

DeleteValue(string byval astrName, long byval allInstanceid)

Class: IUDF
Description:

This will remove a user defined field value.

astrName The name of the UDF for which a value is being deleted.

allInstanceid The id of the specific entity instance for which the UDF is being deleted. (e.g. id for asset #123)

Inputs: byval astrName -
 byval allInstanceid -
Outputs: None
Returns: None

UpdateValue(string byval astrName, long byval allInstanceid, string byval astrValue)

Class: IUDF
Description:

This will update the value of a User Defined Field associated with an entity.

astrName The name of the UDF for which a value is being updated.

allInstanceid The id of the specific entity instance for which the UDF is being updated. (e.g. id for asset #123)

astrValue The value to be updated for this UDF

Inputs: byval astrName -
 byval allInstanceid -
 byval astrValue -
Outputs: None
Returns: None

String Ping()

Class: IUDF
Description:

Return a string indicating whether this object is instantiated.

Inputs: None
Outputs: None
Returns: String

BSEventProc Package Overview**Descripti n****Classes**

IEventProc
IService
IPostSL

Subpackages

None

IEventProc Class**Description**

The IEvent processing interface contains methods used in calls to the Accounting Engine which require the creation of Journal Entries. Information passed to this interface will be combined with data stored in the Accounting Engine to process through Events defined inside the Accounting Engine.

This interface is fundamental to transaction processing between the Operational system and the Accounting Engine.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long DoAssetEvent(Long byval alEventID, long byval alProductid, long byval Assetid, long byref alTransID, ParamArray opt byval ParmArray ())

Class: IEventProc

Description:

Process an Event using an Asset.

Parm Array consists of:

Standard Input Params(variant array) of which there must be at least one (an Asset or Asset Group ID).

Event Modifiers (as many as are applicable). Format as Modifier ID and any inpt parms. Each event modifier is a variant array.

Inputs:

byval alEventID -
byval alProductid -
byval Assetid -
byref alTransID -
opt byval ParmArray () -

Outputs:

None

Returns:

long

String Ping()

Class:

IEventProc

Description:

Return a string indicating whether this object is instantiated.

Inputs:

None

Outputs:

None

Returns:

String

<u>Class:</u>	IEventPr c
<u>Description:</u>	N n
<u>Inputs:</u>	N n
<u>Outputs:</u>	None
<u>Returns:</u>	None

<u>Class:</u>	IService
<u>Description:</u>	N n
<u>Inputs:</u>	N n
<u>Outputs:</u>	None
<u>Returns:</u>	None

ObtainQELineData()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

VerifyBooksets()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

ObtainRuleVars()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

GetDBFieldValue()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

GetJEDetails()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

VerifyBooksets()

<u>Class:</u>	IService
<u>Description:</u>	None
<u>Inputs:</u>	N ne

IPostSL Class**Description**

This is a controller class used to post the Subledgers.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods****Initialize()**

<u>Class:</u>	IPostSL
<u>Description:</u>	
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

Terminate()

<u>Class:</u>	IPostSL
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

long PostAmount(long byval aAssetId, long aQualEventLineID, currency acurTXNAmount, Enum? aEnumAction)

<u>Class:</u>	IPostSL
<u>Description:</u>	None
<u>Inputs:</u>	byval aAssetId - aQualEventLineID - acurTXNAmount - aEnumAction -
<u>Outputs:</u>	None
<u>Returns:</u>	long

CreateSLDetail(long byval SLBalanceID, long byval aJEID, long aProductID, long aBankID, currency acurTXNAmount, string astrDRCRIND, long aCOAID, string astrPostPeriod)

<u>Class:</u>	IPostSL
<u>Description:</u>	None
<u>Inputs:</u>	byval SLBalanceID -

Update a Bookset

It should be possible to update a bookset in the accounting engine. A bookset can not be updated until after it has been viewed.

All of the fields that are on the bookset should be available for update except for the bookset status that will be maintained automatically by the system. The bookset status should always reflect the date the last status changed was processed for this bookset.

Delete a Bookset

It should be possible to delete a bookset. A Bookset can only be deleted if there are no existing Assets associated with the Booksets. A decision needs to be made on how to enforce referential integrity for qualified events that use this bookset.

A bookset can not be deleted until the Bookset details have been viewed.

Issues**Notes**

Delete Product

A Product can only be deleted if there are no existing Assets associated with the Product to be deleted. A product needs to be viewed before it can be deleted.

Use Cas : Rul Maintenance

Scenario: Rule Maintenance

Rules are used to define the calculations required to process an Event. A Rule is similar in nature to an Excel function. Each Rule will expect specific inputs and will have pre-defined, hard-coded behavior. These are complex in nature and will be built with the expectation that an accomplished spreadsheet user is defining the Rules.

Each Rule can be used in many different Qualified Event lines. E.g. it will be possible to define a Rule for Net Receivable that is referred to in several Qualified Event lines. If there was a different net receivable calculation for which was dependant on Product type, there would be two different Rules, Net Receivable for Product A, and Net Receivable for Product B.

Every JE that is processed will need to use a rule to calculate the posting amount.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

Rules are complex in nature. During the startup phases there will be many iterations through Rule management. Once the system is in production, rules will be viewed frequently, but updates will be relatively infrequent.

Input / Output Data:

The Rule will have header information containing: Rule name, Rule Description

Each Rule will also have one or more Rule Lines. The Rule lines will each have all of the data required to perform the calculation. This may include: verbs (SumStream, SubSL, AddAmt, etc.), perands defining an Entity, Parameters, fixed amounts, Accounting periods, Date, and the destination for the result of this calculation. The final destination for every rule needs to be JEPostAmt.

Valid destinations: User variable, JEPostAmt or StreamPostAmts.

- JEPostAmt is the amount of the JE Debit / Credit entry.
- StreamAmts contain the information required to create or update a stream.
- UserVariables are temporary fields defined in this rule, for use on one or more rule lines.

The data used to process rules can come from:

- ParmList - the parameters that are first created in the Parm Definition scenario, and which will then be selected for use when the Business Event, Product is created. These parms will be passed with the call to the Event Processor.
- Constant - a string, number or percent, which is entered and stored with the Rule when the Rule is created.
- User Variable - a variable defined on a previous line of this Rule.

Add a number to subledger sum

GetSLBalance or **SumSLGroup** (as above) -- capture sum in V1
AddAmount V1 amount

Subtract a number from subledger sum

GetSLBalance or **SumSLGroup** (as above) -- capture sum in V1
SubAmount V1 amount1

Sum a series of input numbers

AddAmount amount1, amount2).....

Subtract the sum of a subledger group from the sum of another subledger group

SumSLGroup SLGroupname Accountingperiod -- capture sum in V1
SumSLGroup SLGroupname Accountingperiod -- capture sum in V2
SubAmount V1 V2

-- OR --

DeltaSLGroup SLGroup1 SLGroup2 Accountingperiod

Get balance in first debit subledger of a JE (PMS RDR)

RevDebit JE#

Get balance in first credit subledger of a JE (PMS RCR)

RevCredit JE#

Pass through amount for creating or updating a Stream. This is used to pass a Stream from ATLAS without any special calculations.

StreamAmounts StartDate, MonthArray

Valid Dates / Periods

The Date / Period table will be loaded using SQL, not a GUI. This is not expected to change after the initial implementation.

For Stream verbs the following standard dates are valid:

- Current Date: today's date will be used for processing.
- Start Date: the first date in the Stream.
- End Date: the last date in the stream.

For S/L verbs the following dates are valid:

- Current Period: the current fiscal period will be used for processing.
- Previous Period: the previous period will be used for processing.

There are 4 Type's that are valid to solve the equation:

Parm: the data is passed in, in the Event Parm

Dest: a destination (variable) defined on a previous line of the Rule.

DBField: a field on the database

Constant: a numeric. E.g. 1 or .8

Gui Notes:

Rule

Rule Components

VERB

- ADDAMOUNT
- SIMSTREAM

OPERANDS

- Constants
- Placeholder
- AE FIELD

USER VAR.

ACCTG. kinds

Destinations

- USER VAR
- POSTAMT
- Post Stream

RULE	OPERANDS	DESTINATION
ADDAMOUNT	OP1, OP2	A

- Enable/disable to enforce entry order
- Build grid on fly
- Insert/delete line in grid - Select Right Click
- ~~populate~~ ^{enable} controls as need by verbs

Subtract the sum of a subledger group from the sum of another subledger group

SumSLGroup SLGroupname Accountingperiod Bookset -- capture sum in V1

SumSLGroup SLGroupname Accountingperiod Bookset -- capture sum in V2

SubAmount V1 V2

-- OR --

DeltaSLGroup SLGroup1 SLGroup2 Accountingperiod Bookset

Implementation of the following verbs has been deferred. It is not clear we will need these verbs.

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

SubtractSumSLGroup SLGroupname Accountingperiod Bookset

Defaults:

To simplify the entry of Accounting rules when creating Events the following defaults will be used:

current accounting period

convention that 0 amounts will not be posted

Us Case: AE Maintenance**Scenario: Event Modifier**

The Event Modifier is used to prevent the need for hard-coding special conditions that need to apply to a specific JE. The Event Modifier describes the conditions that must apply before a Qualified Event line evaluates to true.

The Event Modifier will name the exception condition and define how the AE will recognize the data to be evaluated by the Event Processor.

E.g. Fee codes in PMS, may be represented as Event Modifiers.

Role:

The AE administrator enters this information. This information needs to be communicated to I.T. to ensure that the corresponding information is passed from the operational (ATLAS) system.

Any AE user or developer can view this information.

Frequency / Volume:

Definition and Maintenance is primarily a set-up task used by the AE administrator..

The Event Processor will refer to the Event Modifier every time an event is processed. This is a high volume activity.

Input / Output Data:

Every Event modifier will be comprised of one or more event detail lines that will be used when processing this Event Modifier.

Qualified Event Lines will display the Event Modifier by name.

The Event Processor will need to retrieve the Event Modifiers as efficiently as possible. The Event Processor needs to evaluate all lines in the Event Modifier to true before it recognizes the Event Modifier as valid for an Asset. E.g. Event Modifier Sample 1 .

Source	Field	R. Op.	Value
AE	Principal	>	0
StdParmList	LoanType	=	SBA

In this example the Event Processor will check the principal field (in the AE database) to see if it greater than 0, then it will resolve the StdParmList to find loan type and see if the value is SBA. If both of are true then Event Modifier Sample 1 is true.

Header

Event Modifier Name: will be referred to by ATLAS when invoking the Event Processor, as a selectable field on the Qualified Event window, and used by the Event Processor to evaluate Qualified Event lines.

Event Modifier Description**Detail Lines**

Use Case Parm Management

Use Case Rule Management

Business Rules:

It is possible to have 0 parms for the Product Business Event.

The Product Business Event and associated parms will be referenced from outside the AE. Any change to these implies a systems change.

Create Product and Business Event Association

It should be possible to associate any Product in the AE with a Business Event in the AE. It should be possible to add parms to this association, defining the data that will be passed into the AE with the call to the Event Processor.

Since the Event Processor and Qualified Event will both interpret that parameter array by evaluating the position of a parm in the array, it will be necessary to sequence the parameter array to ensure that parameters stay in the correct order.

Remove Association

It should be able to remove the association between any Product and Business Event.

Add Parms

It should be possible to add parms to a Product Business Event at any time. These parms need to be recognized by the system that will call the Event Processor using this Product Business Event

This includes adding a parm at any position of the parm list. This will call for re-sequencing all parameters as required.

Remove Parms

It should be possible to remove a parm from the Product Business Event at any time.

This includes deleting a parm at any position of the parm list. This will call for re-sequencing all parameters as required.

Update Parms

There is no update.

Scenario: Maintaining Qualified Event Lines

After selecting or associating a Business Event Product and creating or updating the Parameter List it will be necessary to maintain the Qualified Event Lines that define this Business Event Product.

This requires the following:

1. Naming the entry (or entries) required.
2. Select an Event Modifier.

Use Case: AE Maintenance

Scenario: Parm Management

Parms are used to define the parameters that will be required to process a Business Event, Event Modifier or Rule. A Parm is a variable name and data type defined to the Accounting Engine and created as a placeholder in the Business Event, Event Modifier and Rule.

This gives us a vehicle for naming and referring to information that will be passed into the AE as part of a call to the Execute Event without hardcoding the signature of every event. Pretty cool.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users and system developers.

Frequency / Volume:

Parms are complex in nature. During the startup phases there will be many iterations through Parm management. Once the system is in production, parms will be viewed frequently, but updates will only be processed when there is an addition or change to an Event call from Atlas to the AE.

Input / Output Data:

Each Parm consists of:

Name: the name of the parameter. This name needs to be easily recognizable so that it can be selected for association with Rules, Business Events and Event Modifiers.

Description: optional

Parameter Type: This identifies the type of data that is being used in this parameter. This can be: String(text or numbers), Numeric(Numbers) or Currency.

Business Rules:

To be defined

Create New Parm

It should be possible to create new Parns for use in the Accounting Engine at any time

- All fields are required, except description.

Viewing a Parm

It should be possible to view any Parm that exists in the accounting engine including all fields associated with the Parm. It will be necessary to look through a long list of parns looking for a Parm that has the functionality required to perform a calculation, without knowing the name of the Parm that is needed.

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
3 May 2001 (03.05.2001)

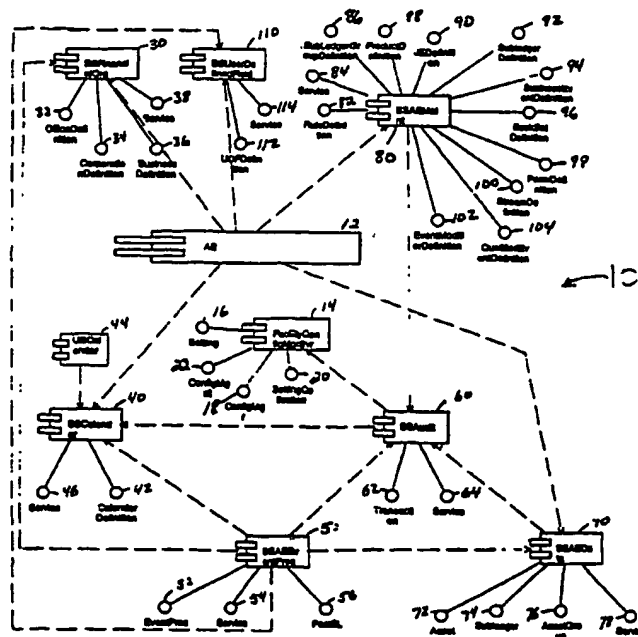
PCT

(10) International Publication Number
WO 01/31482 A2

- (51) International Patent Classification⁷: **G06F 17/00**
- (21) International Application Number: **PCT/US00/29146**
- (22) International Filing Date: **20 October 2000 (20.10.2000)**
- (25) Filing Language: **English**
- (26) Publication Language: **English**
- (30) Priority Data:
09/425,579 22 October 1999 (22.10.1999) **US**
- (71) Applicant: **GENERAL ELECTRIC CAPITAL CORPORATION [US/US];** 44 Old Ridgebury Road, Danbury, CT 06810-5105 (US).
- (72) Inventors: **ARDITTI, Mark, Lee;** 115 Walnut Grove Road, Ridgefield, CT 06877 (US). **WILSON, Judith, A.;** 20 Plumwood Road, Briarcliff Manor, NY 10510 (US). **FAIELLA, Steven;** 55 Mill Plain Road, Unit 22-4, Danbury, CT 06811 (US). **PRICE, David, K.;** 151 Lazy Brook Road, Monroe, CT 06468 (US).
- (74) Agents: **CHASKIN, Jay, L. et al.;** General Electric Company, 3135 Easton Turnpike W3C, Fairfield, CT 06431 (US).
- (81) Designated States (*national*): **AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW.**
- (84) Designated States (*regional*): **ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).**
- Published:**
— Without international search report and to be republished upon receipt of that report.

[Continued on next page]

(54) Title: **LEASE AND LOAN SUB-LEDGER ACCOUNTING METHODS AND SYSTEM**



(57) Abstract: A lease and loan sub-ledger accounting system (10) that provides sub-ledger transaction detail for asset level accounting is described. The accounting system includes a lease and loan accounting engine (12), a plurality of component object model (COMTM) enabled sub-ledger accounting components, and a plurality of programmatic interfaces (140) enabling communication between the accounting components and the accounting engine.

WO 01/31482 A2

package 30 further includes a business definition class 36 containing methods defining a business to accounting engine 12. Financial organization package 30 also includes a service class 38, which provides service to financial organization package 30 by retrieving data organization data from accounting engine 12.

5 Financial organization package 30 and classes described above as well as package and class definitions that follow are described in further technical detail in Appendix A titled Accounting Engine Package Documentation. The descriptions set forth in Appendix B are descriptions of the various accounting functions contained within accounting engine 12.

10 Lease and loan sub-ledger accounting system 10 also includes a calendar package 40 to provide support for multiple fiscal calendars. Calendar package 40 includes a calendar definition class 42 used to identify a fiscal closing date for a bookset and resolve key activity dates used for periodic processes such as bank holidays. If an asset uses multiple booksets, those booksets all use the same
15 calendar. A user service calendar package 44 is also included in calendar package 40 to allow calendar package 40 to run without a complete install of accounting engine 12. A service class 46 which provides service to calendar package 40 is also included.

 Lease and loan sub-ledger accounting system 10 also includes an event
20 processor package 50 to recognize a financial asset such as a piece of equipment, a lease, or a loan to also support account level or asset level accounting. Event processor package 50 includes an event processor class 52 containing methods used to interface with accounting engine 12 that require creation of journal entries and that are fundamental to transaction processing between the operational system and
25 accounting engine 12. A service class 54 is included in event processor package 50 that contains encapsulated retrieval methods for event processor 50. Event processor 50 further contains a post sub-ledger class 56, which is a controller class used to create or modify sub-ledgers and their supporting transaction detail.

© Copyright 1999 General Electric Capital

BSAE Data Packag Overview

Descripti n

Classes

- IAAsset
- IAAssetGroup
- IService
- ISubledger

Subpackages

None

IService Class

Description

This is the service component of the Accounting Engine. This will service: Assets, Subledgers, The Event Processor, and streams.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

String Ping()

<u>Class:</u>	IService
<u>Description:</u>	Return a string indicating whether this object is instantiated.
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	String

ADOR.Recordset GetAllAssetBooksetsByID(long byval alEntityID)

<u>Class:</u>	IService
<u>Description:</u>	Get all of the Booksets associated with an asset by the asset id. Each asset can be used to make entries in multiple booksets.
<u>Inputs:</u>	byval alEntityID -
<u>Outputs:</u>	None
<u>Returns:</u>	ADOR.Recordset

ADOR.Recordset GetAllAssetGroupTypes()

<u>Class:</u>	IService
<u>Description:</u>	Get all of the Asset Group Types in the AE database. Group types are used to identify / stratify the asset groups that have been created. e.g. ATLAS may create a loan Asset Group and a Customer Asset Group. Each of these may have the EntityId 1234567 in ATLAS, since they represent different data. The Ae needs to know what kind of group type (Customer or Loan) to retrieve if asset group value 1234567 is specified.
<u>Inputs:</u>	None
<u>Outputs:</u>	None

ISubledger Class**Description**

This is used to perform additions and updates to the SL Balance and Detail for a single asset. We will ensure that Debits and Credits are written in matching pairs by processing the Debit and Credit using a single invocation of the ISubledger Post method to create both sides of the SL entry.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods****String Ping()****Class:**

ISubledger

Description:

Return a string indicating whether this object is instantiated.

Inputs:

None

Outputs:

None

Returns:

String

Rollover()**Class:**

ISubledger

Description:

This is the method user for year end processing. It will be necessary to close the old year and start and new year. This is considered a S/L account rollover. At the end of a Fiscal year the 12/31 balances are finalized and 1/1 balances are created for the new year. There are many valid reasons the 12/31 balance does not need to = the 1/1 balance. The 1/1 balance may be zero or it may be the total of several other Subledgers that have been rolled in the new 1/1 balance.

Inputs:

None

Outputs:

None

Returns:

None

Long Post(long byval alEntityId, variantarray byval avarPostData)**Class:**

ISubledger

Description:

- Delete from Bookset where SQ_BOOKSET_ID = aiEntityId.
- aiTransNbr = Call LogTransId.

Inputs: byval aiEntityId -
byref aiTransNbr -
Outputs: None
Returns: None

Update(Long byval aiEntityId, string byval astrBooksetName, string byval astrBooksetDesc, integer byval aiActiveId, Long byval aiTaxTypeId, long byval aiReportTypeId, Long byref aiTransNbr, string byval astrDescription)

Class: IBooksetDefinition
Description:

This will update one Bookset in the Accounting Engine.

- Get today's date for adtsStatusDate
- Update Bookset

Inputs: byval aiEntityId -
byval astrBooksetName -
byval astrBooksetDesc -
byval aiActiveId -
byval aiTaxTypeId -
byval aiReportTypeId -
byref aiTransNbr -
byval astrDescription -
Outputs: None
Returns: None

String Ping()

Class: IBooksetDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IProductDefinition Class

Description

This interface contains the methods required to create, update, and use a Product in the Accounting Engine. This will maintain the Product (Lookup table) entity: Product name, description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(String byval astrName, string byval astrDesc, long byref alTransNbr)

Class: IProductDefinition

Description: This will create a Product in the Accounting Engine. Before any Asset can refer to a Product, it will be necessary to define (create) the Product entity.

- Validate required fields: astrName and astrDesc
- Check for duplicate on Product_Name
- Insert into Product_AE
- alTransNbr = Call LogTransId.
- Return ID as long

Inputs: byval astrName -
byval astrDesc -
byref alTransNbr -

Outputs: None

Returns: long

Delete(Long byval alEntityId, long byref alTransNbr)

Class: IProductDefinition

Description: This will delete a Product in the Accounting Engine. Referential integrity needs to be enforced.

- Delete from Product_AE
- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -

Outputs: None

Returns: None

IRuleDefinition Class**Description**

This interface contains the methods required to create, update, and use a Rule in the Accounting Engine. This will maintain the Rule (Lookup table) entity: Rule name, description and the Rule lines that define the Rule.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long CreateHdr(string byval astrRuleName, string byval astrRuleDesc, long byref alTransId)

Class:**IRuleDefinition****Description:**

This will create a Rule in the Accounting Engine. Before an Event can refer to a Rule, it will be necessary to define (create) the Rule entity.

Rule is a Rule header and lines.

Return the Entity Id for the Rule created, not for the Rule lines.

Inputs:**byval astrRuleName -****byval astrRuleDesc -****byref alTransId -****Outputs:****None****Returns:****long**

Long AddRuleLine(Long byval alRuleId, Long byval alVerbLUID, String byval astrRuleDest, Long byval alLineSeq, ADOR.Recordset byval aRSRuleVars, Long byref alTransId)

Class:**IRuleDefinition****Description:**

Add a single Rule line for a Rule. The Rule line is used to define the Rule Verb, Destination and the variables that need to be resolved to process the Rule.

Inputs:**byval alRuleId -****byval alVerbLUID -****byval astrRuleDest -****byval alLineSeq -**

ADOR.Recordset GetJEDetailsById(long byval alEntityId)

Class: IService
Description: This will get the JE header and the debit / credit pairs associated with a single JE in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetQEBooksetsByQELineId(long byval alEntityId)

Class: IService
Description: This will get the Booksets associated with a single Qualified Event line in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetQELinesByPBEt(long byval alProductEntityId, long byval alBusinessEventEntityId)

Class: IService
Description: This will get the Qualified Event Names and Line Sequence numbers associated with a single Event Product.
Inputs: byval alProductEntityId -
byval alBusinessEventEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetRuleLinesByRuleId(long byval alEntityId)

Class: IService
Description: Get a single row from the Rule Field Lookup table using the field Alias.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetRuleById(long byval alEntityId)

Class: IService
Description: This will get a Rule, and all of its associated Rule lines in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

<u>Outputs:</u>	None
<u>Returns:</u>	None

String Ping()

<u>Class:</u>	ISubLedgerGroupDefinition
<u>Description:</u>	Return a string indicating whether this object is instantiated.
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	String

Transaction Class

Description

This interface contains the methods required to create, and use the unique transaction id in the Audit Component. This component may be used by multiple components. This will be recorded on all Accounting Engine entities, when they are created or updated for a complete transaction audit trail. This will also be useful for enabling Undo functionality.

Public enum eAuditTransTypes

```
ecAdd
ecCreate
ecDelete
ecRemove
ecUpdate
end enum
```

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long LogTrans(Long byval aiFacilityId, eAuditTransTypes byval aiTransType, string byval astrEntityName, long byval aiEntityId, Long byref optional aiTransNbr = 0)

Class:

Description:

ITransaction

This will generate a unique transaction number with details about the transaction for audit purposes and return the Entity Id for the transaction generated in this function. The userid will be obtained from the MTS context Object. The date-timestamp will be obtained from the system. The aiTransNbr will be generated if it is zero.

aiTransType The type of database activity that was performed by this transaction as defined in the eAuditTransTypes enum.

astrEntityName The name of entity that is associated with the Entity ID stored for this transaction.

aiEntityId The Entity ID for the entity involved in this activity.

- Get Userid from MTS context Object.
- Get Date-timestamp from system.
- If aiTransNbr = zero

```
tn = create new transaction number
Else
tn = aiTransNbr
```

long CreateRoundingRule(String byval astrName, string byval astrDescription, long byval alRoundingTypeid, integer byval aintRoundingPos)

Class: ICurrencyDefinition
Description: This will create a Currency in the Accounting Engine. Before any Asset can be refer to a Currency, it will be necessary to define (create) the currency and a rounding rule.
Inputs: byval astrName -
 byval astrDescription -
 byval alRoundingTypeid -
 byval aintRoundingPos -
Outputs: None
Returns: long

UpdateCurrency(Long byval alEntityid, VariantArray byval avCurrencyData, long alTransNbr)

Class: ICurrencyDefinition
Description: This will update one Currency Exchange Rate in the Accounting Engine.
Inputs: byval alEntityid -
 byval avCurrencyData -
 alTransNbr -
Outputs: None
Returns: None

UpdateRate(Long byval alEntityid, string astrConversionSource, long byval alConversionRate, long byref alTransNbr)

Class: ICurrencyDefinition
Description: This will update one Currency Exchange Rate in the Accounting Engine.
Inputs: byval alEntityid -
 astrConversionSource -
 byval alConversionRate -
 byref alTransNbr -
Outputs: None
Returns: None

UpdateRoundingRule(Long byval alEntityid, long byval alIndRoundTo, string byval astrRuleDescription, integer byval aintRoundToDecimal, long byref alTransNbr)

Class: ICurrencyDefinition
Description: This will update one Rounding Rule in the Accounting Engine.
Inputs: byval alEntityid -
 byval alIndRoundTo -
 byval astrRuleDescription -
 byval aintRoundToDecimal -
 byref alTransNbr -
Outputs: None

IBusinessDefinition Class**Description**

This interface contains the methods required to define a Business to the Accounting Engine.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long Create(string byval astrBusinessName, string byval astrBusinessDescription, long byref alTransid)

Class:**IBusinessDefinition****Description:**

This will create a business in the Accounting Engine. Before any Office can be defined within a Business, it will be necessary to define (create) the Business entity. Insert the Business

Inputs:

byval astrBusinessName -
byval astrBusinessDescription -
byref alTransid -

Outputs:

None

Returns:

long

Delete(long byval alEntityId, long byref alTransid)

Class:**IBusinessDefinition****Description:**

This will delete a Business from the Accounting Engine. Referential integrity needs to be enforced between this and the Corporation and the office. Delete the business

Inputs:

byval alEntityId -
byref alTransid -

Outputs:

None

Returns:

None

Update(long byval alEntityId, string astrBusinessName, byval astrBusinessDescription, long byref alTransid)

Class:**IBusinessDefinition**

Outputs: byval alBusinessId -
Returns: byval AVCorpId -
 N n
 long

Delete(long byval ALEntityId, long byref alTransID)

Class: IOOfficeDefinition

Description:

This will Delete an Office defined to the Accounting Engine. Referential Integrity needs to be enforced between this and the Corporation and the Asset.

- alTransId = Call LogTrans to log the transaction and get the associated transaction number.
- delete from Office

Inputs: byval ALEntityId -
 byref alTransID -
Outputs: None
Returns: None

RemoveCorp(long byval alOfficeId, byval alCorpId, long byref alTransID)

Class: IOOfficeDefinition

Description:

This will remove a junction relationship between an Office and a Corp.

- Delete from Office_Corp

Inputs: byval alOfficeId -
 byval alCorpId -
 byref alTransID -
Outputs: None
Returns: None

Update(long byval alEntityId, string byval astrOfficeName, string byval astrOfficeDescription, long byval alBusinessId, string byval astrBusinessDescription)

Class: IOOfficeDefinition

Description:

This will update one Office in the Accounting Engine.

- alTransId = Call LogTrans to log the transaction and get the associated transaction number.
- lstrUserId = context.security.getoriginalcaller
- lstrDate = Date
- Update Office

asset group does not exist, the asset group will be created and the assets will be added to it.

Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportAssetGroupUDFs(Long byval astrFileName)

Class: ImportExport
Description: Import Asset Group UDF's from astrFilename. This file will contain asset group name and UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

long ImportAssetUDFs(string byval astrFileName)

Class: ImportExport
Description: Import Asset UDF from astrFilename. This file will contain external asset reference and asset UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: long

Long ImportCurrencyRates(Long byval astrFileName)

Class: ImportExport
Description: Import currency conversion rates from astrFilename. This file will contain currency type, country, rates, and effective dates
Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportCurrencyRoundingRules(Long byval astrFileName)

Class: ImportExport
Description: Import currency conversion rounding rules from astrFilename. This file will contain currency type, country, and rounding rules.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

IEventProc Class**Description**

The IEvent processing interface contains methods used in calls to the Accounting Engine which require the creation of Journal Entries. Information passed to this interface will be combined with data stored in the Accounting Engine to process through Events defined inside the Accounting Engine.

This interface is fundamental to transaction processing between the Operational system and the Accounting Engine.

PublicAccess Attributes**ProtectedAccess Attributes****PrivateAccess Attributes****PublicAccess Methods**

long DoAssetEvent(Long byval alEventID, long byval alProductID, long byval Assetid, long byref alTransID, ParamArray opt byval ParmArray ())

Class: IEventProc

Description:

Process an Event using an Asset.

Parm Array consists of:

Standard Input Params(variant array) of which there must be at least one (an Asset or Asset Group ID).

Event Modifiers (as many as are applicable). Format as Modifier ID and any Inpt parms. Each event modifier is a variant array.

Inputs:

byval alEventID -

byval alProductID -

byval Assetid -

byref alTransID -

opt byval ParmArray () -

Outputs:

None

Returns:

long

String Ping()

Class:

IEventProc

Description:

Return a string indicating whether this object is instantiated.

Inputs:

None

Outputs:

None

Returns:

String

Outputs: None
Returns: None

CheckExistingSubledgerBalance()

Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

GetCorp()

Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

Use Case : Rule Maintenance

Scenario: Rule Maintenance

Rules are used to define the calculations required to process an Event. A Rule is similar in nature to an Excel function. Each Rule will expect specific inputs and will have pre-defined, hard-coded behavior. These are complex in nature and will be built with the expectation that an accomplished spreadsheet user is defining the Rules.

Each Rule can be used in many different Qualified Event lines. E.g. it will be possible to define a Rule for Net Receivable that is referred to in several Qualified Event lines. If there was a different net receivable calculation for which was dependant on Product type, there would be two different Rules, Net Receivable for Product A, and Net Receivable for Product B.

Every JE that is processed will need to use a rule to calculate the posting amount.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

Rules are complex in nature. During the startup phases there will be many iterations through Rule management. Once the system is in production, rules will be viewed frequently, but updates will be relatively infrequent.

Input / Output Data:

The Rule will have header information containing: Rule name, Rule Description

Each Rule will also have one or more Rule Lines. The Rule lines will each have all of the data required to perform the calculation. This may include: verbs (SumStream, SubSL, AddAmt, etc.), operands defining an Entity, Parameters, fixed amounts, Accounting periods, Date, and the destination for the result of this calculation. The final destination for every rule needs to be JEPostAmt.

Valid destinations: User variable, JEPostAmt or StreamPostAmts.

- JEPostAmt is the amount of the JE Debit / Credit entry.
- StreamAmts contain the information required to create or update a stream.
- UserVariables are temporary fields defined in this rule, for use on one or more rule lines.

The data used to process rules can come from:

- ParmList - the parameters that are first created in the Parm Definition scenario, and which will then be selected for use when the Business Event, Product is created. These parms will be passed with the call to the Event Processor.
- Constant - a string, number or percent, which is entered and stored with the Rule when the Rule is created.
- User Variable - a variable defined on a previous line of this Rule.

Us Case: AE Maintenance

Scenario: Event Modifier

The Event Modifier is used to prevent the need for hard-coding special conditions that need to apply to a specific JE. The Event Modifier describes the conditions that must apply before a Qualified Event line evaluates to true.

The Event Modifier will name the exception condition and define how the AE will recognize the data to be evaluated by the Event Processor.

E.g. Fee codes in PMS, may be represented as Event Modifiers.

Role:

The AE administrator enters this information. This information needs to be communicated to I.T. to ensure that the corresponding information is passed from the operational (ATLAS) system.

Any AE user or developer can view this information.

Frequency / Volume:

Definition and Maintenance is primarily a set-up task used by the AE administrator..

The Event Processor will refer to the Event Modifier every time an event is processed. This is a high volume activity.

Input / Output Data:

Every Event modifier will be comprised of one or more event detail lines that will be used when processing this Event Modifier.

Qualified Event Lines will display the Event Modifier by name.

The Event Processor will need to retrieve the Event Modifiers as efficiently as possible. The Event Processor needs to evaluate all lines in the Event Modifier to true before it recognizes the Event Modifier as valid for an Asset. E.g. Event Modifier Sample 1

Source	Field	R. Op.	Value
AE	Principal	>	0
StdParmList	LoanType	=	SBA

In this example the Event Processor will check the principal field (in the AE database) to see if it greater than 0, then it will resolve the StdParmList to find loan type and see if the value is SBA. If both of are true then Event Modifier Sample 1 is true.

Header

Event Modifier Name: will be referred to by ATLAS when invoking the Event Processor, as a selectable field on the Qualified Event window, and used by the Event Processor to evaluate Qualified Event lines.

Event Modifier Description

Detail Lines

Update a Parm

It should be possible to update a Parm in the accounting engine. A Parm can not be updated until after it has been viewed. This includes deleting Parm lines or adding new Parm lines..

All of the fields that are on the Parm should be available for update except for the Parm Name.

Delete a Parm

It should be possible to delete a Parm. A Parm can only be deleted if there are no existing Qualified Events that are associated with the Parm.

A Parm can not be deleted until the details have been viewed.

Issues**Notes**

Have we described enough data types described. The simpler the better.

Use Case: AE Maintenance

Scenario: Subledger Group Definition

Every subledger group is comprised of one or more subledger accounts from the Chart of Accounts. Subledger groups are used to logically connect individual subledger accounts for reference as a group. This will allow us to sum or display a group of subledger accounts that can be referenced by a single meaningful name. E.g. all income accounts or all receivable accounts.

Subledger group definition encompasses two main functions:

1. Adding, deleting, updating and viewing a Subledger Group.
2. Adding, deleting and viewing the Subledger accounts that comprise a subledger group.

Role:

The AE administrator will modify subledger groups. Any AE user can view the subledger group.

The system will use subledger groups for Rules, Journal Entries, and reporting.

Frequency / Volume:

There is moderate setup to define the Subledger groups. Modification or deletion will be infrequent once the AE is setup correctly. Viewing subledger groups will only happen occasionally.

Input / Output Data:

Subledger group name is required and must be unique.

Subledger group description is required.

A list of subledger accounts in the subledger group.

Business Rules:

A subledger group needs to contain at least one subledger account.

All accounts in the group need to be defined on the Chart of Accounts.

It should be possible to add a subledger group or delete a subledger group that is not being referenced in the AE, add a subledger account to a group at any time, remove a subledger account from a group at any time.

Use Case: AE Maintenance

Scenario: Business Event

Business Events are the 'Events' that will be passed into the Accounting Engine Event Processor. These events are used (along with other information) to determine the appropriate set of accounting entries. Events describe a fairly high level process, such as, Booking, Cash Posting, Billing, Terminations, Loan Disposal, etc.

This is primarily a set-up function designed to create the list of Business Events that will be valid in the AE.

The Business Event will be used in combination with a Product and an Event Modifier to define the Qualified Event. The Qualified Event determines the parameters (data passed into the Event Processor with the Event call) that will be necessary to successfully account for this event and the Journal Entries that will ultimately be created.

Role:

AE administrator can add, update or delete a business event. AE users will have a rare need to view the Business Events.

Frequency / Volume:

There will be < 500 business events. This will primarily be an AE setup activity, with occasional use for systems problem solving.

Input / Output Data:

- Business Event Name is required,
- Description is optional.

Business Rules:

Business Event Name must be unique.

Business Events that are used in calls to the AE are required for successful completion of the desired accounting transaction. If the Business Event does not exist in a Product Business Event association the calling system will be expected to reject the entire transaction to maintain the integrity of the accounting and operational data.

Create

It should be possible to create a Business Event at any time. It will be necessary to communicate the new business event to systems personnel to ensure that calls to the AE recognize this as a valid event.

View

The list of all valid Business Events needs to be viewable as part of the Associate Product, Business Event and Parm use case.

Use Case: Stream Maintenance

Scenario: Streams

The Accounting Engine needs to store streams of data, which represent a series of monthly amounts over time. Accounting Engine Maintenance will define the Name and Description for each stream that needs to exist.

At the current time we expect to use Fees and Expense Streams. This remains to be decided.

Role:

AE administrator will be responsible for defining the Streams that may be used for a Loan. AE users may occasionally view this information.

The AE system will only use this information for display or reporting.

Frequency / Volume:

Stream definition and viewing is very infrequent.

Stream display will be more occasional. There will be very few types of Streams.

Input / Output Data:

Stream Name: required.

Stream Description: optional.

Business Rules:

Streams will need to be defined before they can be passed into the AE from ATLAS or used by the AE in a Rule.

Update a Stream

Any field may be updated.

Delete a Stream

Referential integrity needs to be enforced to the Asset Stream.

Create a Stream

A Stream can be created at any time.

Tables

Asset_Stream_Description

Referential integrity to: Asset Stream.

Issue :

What do we do to ensure referential integrity to rules referring to a Stream that is deleted? This seems like it will need to be programmatic. Perhaps we should eliminate the concept of deleting a Stream type. Let's get real, when are we going to delete a Stream type.

Use Case: Calendar

Scenario: Defining and using the Calendar

The Calendar component is designed as a re-useable component that encapsulates calendar functionality.

This component needs to fill three main needs:

- The calendar needs to be capable of defining and mapping a user defined activity to a day on the calendar.
- It should provide the methods required for ATLAS and the Accounting Engine to identify a pre-specified set of dates, relative to any day on the calendar. E.g. previous fiscal year end or day of the week.
- It should provide the fiscal views of the calendar that are required for the Accounting Engine. The calendar needs to be able to:
 - Identify the current fiscal month. This is used for the majority of accounting entries.
 - Identify the fiscal month (and year) for any given day.
 - Identify Calendar month-end, or any other Activity date.
 - Identify the day that fiscal processing should occur for every month in the year.

Role:

The AE administrator maintains the Calendar. The calendar is used extensively by the AE system.

ATLAS will use the calendar component in the billing component.

Frequency / Volume:

The base calendar will be created at the time the system is set-up. There will be very few calendars (<3)

Maintenance for the Accounting Engine should be once a year, to add one more year to the calendar. A year will have a row for every day of the year in the date table.

Calendar maintenance may occur more frequently if the Calendar component is re-used by ATLAS to enhance scheduling, interest calculations, or some other time dependant function.

There will be very few Calendar Activity Types. (<25).

The calendar business services will be referenced for many asynchronous processes. e.g. Billing, accruals, Accounting events.

Related Scenarios:

Defining and maintaining Calendar Activity Types.

Calendar Activity Dates.

Getting dates relative to a date.

Input / Output Data:

There are three basic pieces of information that make a calendar meaningful for the accounting engine:

- Calendar header information: Name and Description that can be used to identify which calendar is being used by an Asset or Bookset. Last update date, last updated by, current fiscal month and current fiscal year are also stored with the calendar.
- Calendar days: Physical day and fiscal month. It should be possible to retrieve information for every day of the year.
- Calendar Activity Type: a unique Activity Name and any Description. E.g. Calendar month end, fiscal month end. A special reserved indicator will be created to indicate that some activity types need to exist for the AE to successfully process entries and can not be updated or deleted. Activity Type Name and description is required.

Creating the Calendar

A base calendar containing the Calendar name and description needs to be created once for every calendar that will be used in the calendar component.

It will be necessary to add one year of days, in full year increments, to the calendar. At the time the calendar days are created, the user will specify the fiscal month, by starting date of month, for the entire year. This should be implemented as a simple, easy to understand GUI. Since this is done once a year, it is important that this is intuitive and easy to use. A reasonable processing delay (1 minute) is acceptable.

The system should update the user-id and update date.

The accrual process will need to change the current fiscal month and year.

Deleting the Calendar

It will not be possible to delete an entire calendar. This seems both complex and unlikely to be used.

It should be possible to delete (purge) a range of dates as part of standard maintenance.

It should be possible to delete the association between a calendar activity and a day.

Updating the Calendar

It is possible to update the calendar activity type name and description. The system should update the user-id and update date. The fiscal accrual process should update the fiscal month and year.

It is possible to update the days in a calendar by changing the fiscal start month for one or more periods. This will need to change every day in the year to reflect the current fiscal period.

The first day of each fiscal year should be 1/1.

Each fiscal period must have a fiscal start date that is less than the date of the following fiscal period. E.g. If fiscal March starts on 2/28/2000, then fiscal February must start after 1/1/2000 and before 2/28/2000.

Scenario: Defining and maintaining Calendar Activity Types

The Calendar will need the ability to associate activities with dates. The first step in creating this association will be the creation of the Calendar Activity Types.

The second step will be associating the dates in the Calendar with Activity Types that have already been created and will be documented in the Calendar Activity Dates scenario.

Input / Output Data:

- Calendar Activity Type Name and Description are the only two fields that can be created and maintained for the Calendar Activity Type.
- Last Update, Updated by and Reserved Indicator will be generated by the system. The system will always set the Reserved Indicator off.
 - Reserved Activity Types need to be created by I.T. personnel.

Creating Calendar Activity Types

Calendar Activity Types need to be created once for use by all calendars that will be defined in the calendar component.

Calendar Activity Types will not be viewed or changed very often. The most frequent use of Calendar Activity Types will be viewing the Calendar Activity Type associated with a given day.

Calendar Activities can be 'reserved' for system use by setting an indicator on the Calendar Activity Type table. This can only be done by an I.T. system administrator role using SQL or database tools.

Calendar Activity Type Name needs to be unique.

~~The system should update the user id and update date.~~

Deleting Calendar Activity Types

It will be possible to delete a Calendar Activity Type that is not associated with any Activity Date. Attempting to delete a Calendar Activity Type that is used by an existing Calendar Activity Date will cause referential integrity error.

It is not possible to delete a Reserved Activity Type.

Updating Calendar Activity Types

It will be possible to update Calendar Activity Type Name and Description for activities that are not reserved.

It is not possible to update a Reserved Activity Type.

Viewing Calendar Activity Types

It should be possible to view the following information for a calendar:

- A list of all Activity Types: Name and Description.
- A list of all Reserved Activity Types: Name and Description.

Scenario: Getting dates relative to a date

The Calendar component will be required to provide the methods required for ATLAS and the Accounting Engine to identify a pre-specified set of dates, relative to any day of the year. This is a system requirement, there is no User Interface required.

Input / Output Data:**Input:**

- Date, optional, byref. This will default to today's date if it is not entered.
 - Note: there will be times when the system (ATLAS or AE) will need data based on another point in time. By allowing the date as an input field we ensure that we can create the date view for any point in time.
 - Note: this will also ensure that date errors are not generated due to time differences on the servers used to run various applications or components.

Outputs:

Every input request that is successfully processed needs to generate all of the following outputs relative to the input date.

- Today's date: the calendar date for today as obtained from the Calendar application server.
- Input date: this is returning the date that was passed in the input request. This will default to today if the date is not passed as an input argument.
- Day number: The day of the month for the input date.
- Day of the Week: Mon., Tues., etc. expressed as an integer.
 - 1 Sunday, 2 Monday, 3 Tuesday, 4 Wednesday, 5 Thursday, 6 Friday, 7 Saturday
- Day of the Week #: This is used to identify whether the day of the week is the 1st, 2nd, 3rd, 4th or 5th occurrence in the month..
- Last occurrence in month: Boolean. This will be true if this is the last time this day of the week will occur in this month.
- Two weeks ago: The input date minus 14 days, returned as a full date.
- One month ago: The input date minus one month. If the day referenced in the input date is greater than the highest day in the previous month, then this will return the highest day in the previous month. e.g. if the input date is 2/27/99, the return date will be 1/27/99. If the input date is 12/31/99, the return date will be 11/30/99.
- Two Months ago: The input date minus two months. If the day referenced in the input date is greater than the highest day in the month before the previous month, then this will return the highest day in the month before the previous month. e.g. if the input date is 2/27/99, the return date will be 12/27/98. If the input date is 08/31/99, the return date will be 06/30/99.

Tables: None

Us Case: UDF**Scenario: Maintain UDF Name and Description**

User Defined Fields (UDF's) can be created in the UDF component for reporting and inquiry. UDF's will be created to improve the flexibility of the Accounting Engine. This will allow storing data that is not core accounting data, but which is useful in combination with the accounting data. This will reduce the amount of maintenance required in the AE to accommodate special cases, which require non-financial data.

It is necessary to perform standard maintenance on a User Defined Field to Add, Update, and Delete the UDF Name, Description and Table Name.

Every UDF will identify one table on the database, which will be used to resolve the UDF to a specific Asset, Asset Group or other instance on an existing table. The table name will be stored with the UDF Name and description to ensure that all instances used by a single UDF Name are resolved using the same table.

UDF Maintenance will be performed by a call to the AE from the ATLAS system.

UDF's should be carefully implemented to prevent the proliferation of UDF's in the AE. This is a relatively inefficient structure. It should be used to enhance flexibility, not as an extensive reporting aid.

Role:

ATLAS system will be the actor in the Maintain UDF use case.

Frequency / Volume:

This maintenance will be rare. There will only be a handful of UDF's used in the initial SBF implementation of the AE.

Input / Output Data:

User Defined Field Name: required and unique.

User Defined Field Description: optional.

Table Name: required. This needs to identify an existing table accessible by the AE.

Business Rules:

Do not delete UDF Name that is being referenced by a UDF name/ value pair.

Tables:

UDF_Names: It should be possible to retrieve the entire list of valid names.

UDF Values

<u>UDF Name</u>	<u>UDF Value</u>	<u>InstanceId</u>
Salesman	Jane Doe	AG 1234
Salesman	John Doe	AG 3456

- Every office may be in multiple corporations
- Every corporation may have multiple offices

Scenario: Using the Financial Organization

Every loan is associated with a corporation, a business and an office. This association is maintained for the life of the loan to ensure that loan accounting and reporting are both correct with respect to the financial organization. Any change in the Office or Corporation associated with a loan implies a set of *transfer* accounting entries. A change in the business does not impact the underlying accounting.

Role:

ATLAS user role will pass the association between a loan (asset) and the corporation and office that will be used for accounting purposes when an asset is created.

? where will an office transfer be originated?

Frequency / Volume:

These are transactions scattered throughout the day. In the event of a portfolio acquisition there may be batch processes used to load or create hundreds of these in a short period of time.

Input / Output Data:

The association of the Loan, Corporation and Office are best defined as part of the interface document used to define how to create an asset in the Accounting Engine. At a minimum the AE will require the following inputs:

Loan number, Corporation, Office and an external asset reference number.

The accounting engine will be required to return an asset number to ATLAS to identify the accounting asset.

Business Rules:

Any change in the corporation or office associated with the loan (financial asset) requires office transfer or corp. transfer accounting entries. Maintenance of office and corporation in ATLAS and the AE must be treated as a tightly coordinated process.

It is necessary to identify the month ending balance associated with an asset for every month in the current year.

It should also be possible to answer the following questions related to subledgers:

1. What is the subledger balance, for a single asset, for a specific month.
2. What is the sum of the balances for a subledger account, for all of the assets in an asset group, for a specific month.
3. What are the month ending balances, by month, for all of the assets in a specific year.
4. What are the Subledger transaction details for a given period of time for a Subledger account.
5. What Subledger balances exist for an asset for a given accounting period.

Issues

1. We have deferred the implementation of using Asset Groups for Subledgers.

Use Case: Subledgers

Scenario: Subledger Group, Subledger Transaction Processing

A Subledger Group is a group of logically related Subledgers that have semantic meaning to the business. E.g. the list of income subledgers might be called the IncomeSubledgerGroup, and the list of receivable subledgers might be referred to as the ReceivableSubledgerGroup. This allows the group to be acted or reported upon in concert, without having to specifically name each subledger to perform the action or create the report.

Role:

The system role can see or refer to any valid subledger group.

Frequency / Volume:

Subledger groups will be referenced frequently by the Event Processor, inquiries and reporting.

Estimate between 10 and 250 Subledger groups. A typical subledger group will contain about 5 – 10 subledgers. It is unlikely that a subledger group will have more than 20 subledgers. It must have at least one subledger.

Input / Output Data:

Subledger group name, description, subledgers associated with the subledger group.

A subledger can be associated with many Subledger groups.

Business Rules:

Every Subledger group will include at least one subledger account.

Actions

It should be possible to retrieve information about subledger groups to answer all of the following questions:

1. What is the sum of the balances for the subledger accounts in a subledger group, for a single asset, for a specific month.
2. What is the sum of the balances for the subledger accounts in a subledger group, for all of the assets in an asset group, for a specific month.
3. What is the sum of the month ending balances, by month, for all of the assets in a subledger group for a specific year.
4. What are the Subledger transaction details for a given period of time for every Subledger account in the Subledger group.
5. What Subledger balances exist for an asset for a given accounting period.

Use Case: Asset**Scenario: Use an Asset**

An Asset can represent a physical piece of equipment or a financial entity such as a loan or an unapplied cash account. All Assets will have a corresponding asset represented on the source (ATLAS) system. The ATLAS representation may be a loan or it may be suspense account.

Since the Asset does not contain any financial information the accounting engine will only store the current representation of the Asset. All of the financial information for an asset will be stored in Subledgers or Streams.

Role:

System actors, not human actors, use assets. ATLAS and the AE will both need to use Assets.

Frequency / Volume:

There will be a low to moderate volume activity against the Asset. It will be referenced primarily at the time of booking, for Subledger inquiries and reporting.

Input / Output Data:

The asset contains the following:

Office / Corp : Office and Corp need to be validated as a valid Office / Corp. combination.

Facility: the facility (external system, ATLAS) that combined with the external asset reference will be unique.

External-asset reference: This is the way ATLAS refers to this asset. This is an important cross-reference point for ensuring the integrity of ATLAS to the AE.

Currency: US \$ will be used for SBF implementation.

Permanent asset id: this is used to ensure that we can refer back to the original asset in the event it is necessary to book this deal with a new Loan number.

Volume Ind: is this account new volume for this month? Should this be a volume date?

Status date: date of the last status update.

Asset status: Active, Inactive, Pre-book.

Earning status date: date of the last earning status update.

Earning Status: Earning, Non-earning.

Business Rules:

It is not possible to physically delete an Asset. Asset will be logically deleted by marking them inactive and populating the Status date.

Use Case: Asset Group**Scenario: Use an Asset Group**

An Asset Group is used to identify multiple assets as related entities. This relationship can define things as diverse as assets on an Account, Customer or Vendor. It will be possible to inquire or report against various pieces of data in the AE by specifying the Asset Group. E.g. The AE will be able to return the subledger balance for an asset group by summing a single subledger account for all assets in the asset group.

Role:

System actors, not human actors, use assets groups. ATLAS and the AE will both need to use Assets groups. This will be very useful for reporting and online inquiries.

Frequency / Volume:

There will be a low to moderate volume activity against the Asset group. It will be referenced primarily for Customer inquiries, Subledger inquiries and reporting.

Input / Output Data:

The asset group contains the following:

Group type: an identifier to specify why these assets are grouped together. Thus it will be possible to have one group which is used to represent a customer, and another for a portfolio being serviced.

External Group Reference: the reference (name, description or id) the external system (ATLAS) uses to refer to this asset.

Business Rules:**Actions**

From the external (ATLAS) system it should be possible to:

- Add a new asset group.
- Delete an asset group.
- Add assets to an existing asset group
- Remove assets from an existing asset group.
- It will not be possible to update an asset group.

From the AE or from ATLAS it should be possible to view:

- All of the asset detail for the assets associated with an asset group.
- The subledger detail for all of the assets in the asset group for a date range.

Use Case: Processing an Event

~~Scenario: ExecuteAsset, ExecuteAssetGroup~~

The Event Processor will be used to post all debits and credits in the Accounting Engine. The Event Processor will also be responsible for maintaining any streams that are used in the AE.

The Event Processor will have a public interface that can be invoked by ATLAS, or an AE internal process. Each invocation will identify the Business Event being processed and pass in any information required to correctly process the Event.

The Event Processor will iterate through the Qualified Event Lines for a Business Product Event, identifying the required Journal Entries by resolving the Event Modifiers for each line, and then processing the Rules required to perform the calculations needed to successfully create the posting amount.

The Event Processor can be invoked to process an Event for a single Asset or for an Asset Group.

~~Every successful ExecuteAssetEvent will return a unique transaction id to the invoking system unless the invoking system passes in the transaction id of a previous transaction to ensure they look like a single transaction in the AE.~~

Role:

This is a system actor. There is no direct human Actor.

Frequency / Volume:

This is used for every single debit and credit posted in the AE. This is high volume, time critical process.

Input / Output Data:

Business Event

Product

Asset ID

Unique Transaction ID

Business Product Event standard parameters

Business Rules:

See Rules for Verb list.

Processing

Technical look:

ExecuteAsset (byval astrEvent : string, byval alProductId? : long, byval Assetid : long, byref alTransID : long, byval StdParmList : VariantArray)

ExecuteAssetGroup (byval astrEvent : string, byval alProductId? : long, byval alAsseGroupid : long, byref alTransID : long, byval StdParmList : VariantArray)

Validity checks

- **Lookup / Verify the Business Event – Product combination. This needs to be validated in two places:**
 - **The Business Event – Product table which is used to resolve the StdParmList**
 - **The Qualified Event table to ensure that there is at least one Qualified Event Line for this Business Event – Product.**
- **Verify the ParmS passed in the Standard Parm list (VariantArray) against the parameters expected by the Business Event Product.**
 - **Have the correct number of parms been passed in to the AE. The number of parms must = the UpperBounds of the EventProdParm.**
 - **Are the parms of the correct type; currency, string, etc**
- **Verify Business Event Product ParmS against the parameters expected by the Event Modifier.**
 - **Event Modifier exists**
 - **ParmS are present and of the correct type.**
- **Resolve Bookset to Asset Bookset to identify the Qualified Event Lines that are using booksets that may be used to make Journal Entries for this Asset. It is not necessary to process QE Lines that do not have a Booksets valid for this Asset.**
- **Check validity of the Asset being used to process this event.**

Qualified Event Processing

- **Obtain Qualified Event Lines.**
 - **Sort by Entry Name and Priority.**

For Each QE Line, until a match has been found for this Entry Name:

- **IF line has an Event Modifier**
 - **Has this Event Modifier been checked already? If so, then use the results of the last check to save processing time, if not Check Event Modifier conditions**
 - **Save the results of the Event Modifier evaluation so we know if the EM is True, False or not evaluated yet.**
 - **IF they don't match, skip to next line**

Process one Qualified Event Line

- **Grab Rule**
 - **Execute SQL select(s) to obtain ?? data (or obtain info from existing RS. E.g. Asset_AE to get Principal Balance.**
 - **Execute Rule using:**
 - **StdParmList**

APPENDIX C FORM INTERFACE DEFINITIONS

© Copyright 1999 General Electric Capital

LOGICAL VIEW REPORT

SELECTED LOGICAL VIEW REPORT

Bookset

FrmUDFMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 2505

cFORM_MIN_WIDTH : = 7395

gbUpdateUDF : Boolean

gbAddNewUDF : Boolean

Public Operations:

newData () :

deleteData () :

cancelUDF () :

=

METHOD : cancelUDF

PURPOSE : This will cancel an add to the udf Record set

PARMS :

RETURN :

=

deleteUDF () :

=

METHOD : deleteUDF

PURPOSE : This will delete a record from the udf Table

PARMS :

RETURN :

=

missingData () : Boolean

=

METHOD : missingData

PURPOSE : This method will determine if any required fields are missing

PARMS :

RETURN : lbBad As Boolean

=

saveUDF () :

=

METHOD : saveUDF

PURPOSE : This will determine if the data should inserted or updated.

The appropriate call to the business service will be made

PARMS :

LOGICAL VIEW REPORT

dgrdUDF_RowColChange (LastRow : Variant, LastCol : Integer) :
 dgrdUDF_PostEvent (MsgId : Integer) :
 dgrdUDF_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
 dgrdUDF_BeforeRowColChange (Cancel : Integer) :
 abarUDF_Click (Tool : ActiveBarLibraryCU.Tool) :

MDImain
Product
frmBusinessAdd

Public Operations:

missingData () : Boolean

Private Operations:

Form_Unload (Cancel : Integer) :
 Form_Load () :

frmBusinessMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3555
 cFORM_MIN_WIDTH : = 8670
 gbUpdateBusiness : Boolean
 gbAddNewBusiness : Boolean

Public Operations:

deleteData () :
 addNewBusiness () :
 newData () :

Private Operations:

disableText () :

=

METHOD : disableText
PURPOSE : This will disable the text entry areas
PARMS :

RETURN : None

=

enableText () :

=

METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

LOGICAL VIEW REPORT

saveBusiness () :

=

METHOD : saveBusiness**PURPOSE :** This method will determine if it should create or update the record.

The method will then call the appropriate method and request the service from the Business Service Layer

PARMS :**RETURN :**

=

setTextFields () :

=

METHOD : setTextFields**PURPOSE :** This method will bind the database fields to the text fields**PARMS :****RETURN :**

=

getAllBusinessData () :

=

METHOD : getAllBusinessData**PURPOSE :** This event will retrieve all the Business data.**PARMS :****RETURN :**

=

Form_Load () :**dgrdBuiness_RowColChange (LastRow : Variant, LastCol : Integer) :**

=

METHOD : dgrdBuiness_RowColChange**PURPOSE :** This event will post an event if the prior row was saved**PARMS :****RETURN :**

=

dgrdBuiness_PostEvent (MsgId : Integer) :

=

METHOD : dgrdBuiness_PostEvent**PURPOSE :** This event will be triggered after the PostMsg. It is here where the grid will be refreshed.**PARMS :****RETURN :**

=

dgrdBuiness_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

-C5-

LOGICAL VIEW REPORT

Public Operations:**resetAllControlsInError () :**

=

METHOD : resetAllControlsInError**PURPOSE :** This method will reset the controls**PARMS :****RETURN :** None

=

newData () :**PARMS :****RETURN :** None

=

Private Operations:**validateActivity () : Boolean**

=

METHOD : validateActivity**PURPOSE :** this method will validate the data that was entered**PARMS :****RETURN :** Boolean

=

sndPanelMsg (aMsgType : panelMsg) :

=

METHOD : sndPanelMsg**PURPOSE :** This will send the appropriate message to the panels status**PARMS :**

aMsgType (panelMsg) =

RETURN : None

=

ValidateForm (aAction : frmCalendarRSActions) : Boolean

=

METHOD : ValidateForm**PURPOSE :** check each recordset that can be updated to see if it has changed.

if it has changed, then validate the data entered

if validations are passed, then call the appropriate save routine

PARMS :

aAction (frmCalendarRSActions) =

RETURN : Long

=

processRS_Calendar (aAction : frmCalendarRSActions, svParms() : Variant) : Long

=

METHOD : processRS_Calendar

LOGICAL VIEW REPORT

RETURN : None

=

PopulateActivityTypes 0 :

=

METHOD : PopulateActivityTypes

PURPOSE : load the activity drop down used to select an activity type to associate with a date.

this dropdown is only used to add a new activity / activity date association

PARMS :

RETURN : None

=

PopulateCalendars 0 :

=

METHOD : PopulateCalendars

PURPOSE : load the calendar grid with all the calendars defined

PARMS :

RETURN : None

=

ActivitiesForOneYear 0 :

=

METHOD : ActivitiesForOneYear

PURPOSE : load up the activities for the calendar and calendar year selected.

PARMS :

RETURN : None

=

AllYears 0 :

=

METHOD : AllYears

PURPOSE : load the year dropdown box, default value is the current year

PARMS :

RETURN : None

=

AddNewYear 0 :

=

METHOD : AddNewYear

PURPOSE : add a year to the year dropdown used to select the year being viewed / updated

PARMS :

RETURN : None

=

LOGICAL VIEW REPORT

 =
cboYear_Click () :

 =
METHOD : cboYear_Click
PURPOSE : check to see if an update has been done, then sync details
PARMS :
RETURN : None

 =
Form_Unload (Cancel : Integer) :

 =
METHOD : Form_Unload
PURPOSE : set the globals to nothing
PARMS :
 Cancel [Integer] =
RETURN : None

 =
Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :

PARMS :
 Cancel [Integer] =
 UnloadMode [Integer] =
RETURN : None

 =
dgrdAllCalendars_PostEvent (MsgId : Integer) :

 =
METHOD : dgrdAllCalendars_PostEvent
PURPOSE : This method will refresh the datasets and the grids
PARMS :
 MsgId [Integer] =
RETURN : None

 =
dgrdAllCalendars_BeforeRowColChange (Cancel : Integer) :

PARMS :
 Cancel [Integer] =
RETURN : None

 =
dgrdActivities_RowColChange (LastRow : Variant, LastCol : Integer) :

PARMS :
 LastRow [Variant] =
 LastCol [Integer] =
RETURN : None

 =
dgrdActivities_PostEvent (MsgId : Integer) :

 =
METHOD : dgrdActivities_PostEvent
PURPOSE : The Post Event process will refresh the record set for the grid

LOGICAL VIEW REPORT

Private Attributes:

cFORM_MIN_HEIGHT : = 6615

cFORM_MIN_WIDTH : = 8835

Some Form Constants

Private Operations:

ValidateForm (aAction : frmActivityTypeRSActions) : Long

=

METHOD : ValidateForm

PURPOSE : validate the data that has changed, if any

PARMS :

aAction [frmActivityTypeRSActions] =

RETURN : Long

=

processRS_ActivityTypes (aAction : frmActivityTypeRSActions, avParms() : Variant) : Long

=

METHOD : processRS_ActivityTypes

PURPOSE : determine if there are any changes to the recordset. If there are any changes then edit the cahgnes and save the RS.

This also controls the button enable/disable logic

PARMS :

aAction [frmActivityTypeRSActions] =

avParms [Variant]. =

RETURN : Long

=

Form_Unload (Cancel : Integer) :

=

METHOD : Form_Unload

PURPOSE : form, form, go away.
use the recordsets another day

PARMS :

Cancel [Integer] =

RETURN : None

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :

=

METHOD : Form_QueryUnload

PURPOSE : preprocessing for form unload.
save the RS if necessary

PARMS :

Cancel [Integer] =

UnloadMode [Integer] =

RETURN : None

=

populate_ReservedActivityTypes 0 :

=

LOGICAL VIEW REPORT

Cancel [Integer] =
RETURN : None

abarPopupMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

METHOD : abarPopupMenu_Click
PURPOSE : 'This controls the behaviour of the ActiveBar menu control.
delete: delete the row and select the first row in the recordset.
add: add a new row to the recordset.
PARMS :
Tool [ActiveBarLibraryCtl.Tool] =
RETURN : None

frmCurrFiscalPeriod

MODULE : frmCurrFiscalPeriod
PURPOSE : this form is used to update the current fiscal period for any calendar defined in the AE.

Private Attributes:

cFORM_MIN_HEIGHT : = 2460
cFORM_MIN_WIDTH : = 9690
Some Form Constants
glCalendarID : Long

Public Operations:

CalendarID (alCalendarId : Long) :

METHOD : CalendarID
PURPOSE : This is the setter method for the CalendarId
PARMS :
alCalendarId [Long] =
RETURN : None

CalendarID 0 : Long

METHOD : CalendarID
PURPOSE : This is a getter method for the CalendarId Property
PARMS :
RETURN : Long

LOGICAL VIEW REPORT

Public Operations:

cancelAdd 0 :
deleteData 0 :
newData 0 :

Private Operations:

ClearSearchFields 0 :
ManageSearchButton 0 :
EnableEditFields (bEnable : Boolean) :
ProcessRS_EventMeds (aAction : frmEventModActions, avParms() : Variant) : frmEventModRC
cmdSearch_Click 0 :

cmdClear_Click 0 :
txtModifierName_Change 0 :
txtModifierDescrip_Change 0 :
Form_Unload (Cancel : Integer) :
Form_Load 0 :
dgrdEventMeds_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdEventMeds_PostEvent (MsgId : Integer) :
dgrdEventMeds_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdEventMeds_BeforeRowColChange (Cancel : Integer) :
abarPopupMenu_Click (Tool : ActiveBarLibraryCtlTool) :

frmFiscalPeriodStartDates

Private Attributes:

cActivityTypeFiscalStart : = 1
cFORM_MIN_HEIGHT : = 3500
cFORM_MIN_WIDTH : = 8500
Some Form Constants
giCalendarID : Long
gstrCalendarName : String
Dim globals, private to form

Public Operations:

CalendarID (aiCalendarId : Long) :
CalendarName (astrCalendarName : String) :

Private Operations:

ValidateForm (aAction : frmCalendarRSActions) : Long

=
METHOD : ValidateForm
PURPOSE : check each recordset that can be updated to see if it has changed.
if it has changed, then validate the data entered
if validations are passed, then call the appropriate save routine
PARMS :
aAction [frmCalendarRSActions] =
RETURN : Long

=

LOGICAL VIEW REPORT

AllYears () :

=

METHOD : AllYears**PURPOSE :** load the year dropdown box, default value is the current year**PARMS :****RETURN :** None

=

AddNewYear () :

=

METHOD : AddNewYear**PURPOSE :** add a year to the year dropdown used to select the year being viewed / updated**PARMS :****RETURN :** None

=

Form_Load () :

=

METHOD : Form_Load**PURPOSE :** load the form'

dim the local variables, set the clientside cursor,

PARMS :**RETURN :** None

=

cboYear_Click () :

=

METHOD : cboYear_Click**PURPOSE :** check to see if an update has been done, then sync details**PARMS :****RETURN :** None

=

pvdFiscalMonth_Validate (Index : Integer, Cancel : Boolean) :

=

METHOD : pvdFiscalMonth_Validate**PURPOSE :** if the fiscal period is changing move the data to the RS for update.

the RS is indexed relative to 1, the pvdFiscalMonth array is indexed relative to 0

PARMS :

Index [Integer] =

Cancel [Boolean] =

RETURN : None

=

Form_Unload (Cancel : Integer) :

=

LOGICAL VIEW REPORT

abarContextMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

=

METHOD : abarContextMenu_Click

PURPOSE : This sub will be fired when there is a click on any of the context menu's for this

form. Identify the appropriate click and take the appropriate action.

PARMS :

Tool [ActiveBarLibraryCtl.Tool] =

RETURN : None

=

frmJEMaintenance

Private Attributes:

cFORM_MIN_HEIGHT : = 7140

cFORM_MIN_WIDTH : = 10695

Some Form Constants

ghJEChange : Boolean

gJcID : Long

Public Operations:

addNewDRCRPair () :

deleteData () :

newData () :

cancelDrCr () :

cancelJe () :

=

METHOD : cancelJe

PURPOSE : This method will cancel a new record for JE Header pairs after the user clicked cancel from the context menu. It will set the context menu and enable or disable all fields

PARMS :

RETURN :

=

save () :

Private Sub txtJENumber_KeyUp(KeyCode As Integer, Shift As Integer)

PURPOSE : Check to insure that some search criteria have been entered.

If not, disable the search button

If Len(Trim(txtJENumber.Text)) < 3 And Trim(txtJENumber.Text) = "" Then

cmdSearch.Enabled = False

Else

cmdSearch.Enabled = True

End If

End Sub

LOGICAL VIEW REPORT

Private Operations:

```

enableDRCRGrid (bEnable : Boolean) :
ClearSearchFields () :
ManageSearchButton () :
EnableEditFields (bEnable : Boolean) :
txtJENumber_KeyPress (KeyAscii : Integer) :
txtJENName_KeyPress (KeyAscii : Integer) :
txtUDEscrip_KeyPress (KeyAscii : Integer) :
Reset_Errors () :
*****
*****
populate_JE_Headers (aInitGrid : Boolean = False) :
*****
*****
processRS_SLCOA (aAction : frmJeRSActions, avParms() : Variant) : Long
*****
*****
processRS_DRCRPairs (aAction : frmJeRSActions, avParms() : Variant) : Long
processRS_JEHeaders (aAction : frmJeRSActions, avParms() : Variant) : Long
*****
*****

```

METHOD : processRS_JEHeaders

PURPOSE : Any Movement(or add/delete) in the J/E Header Recordset is processed here. Before we process the change, we check to see if we need to save the information associated with the currently active row. Before saving any information, the content of the columns is validated. This method will also make calls to child recordset processes of the same type.

PARMS :

aAction [frmJeRSActions] = see Enum in GenDecs for details

avParms [Variant] = if any parms need to be passed in

RETURN : Long (0=success, -1=failure)

```

=
txtJENumber_KeyPress (KeyAscii : Integer) :
txtJENumber_Change () :
txtJENName_Change () :
tdbcDebitSL_ItemChange () :
tdbcCreditSL_ItemChange () :
Form_Unload (Cancel : Integer) :
dgrdJournalEntries_RowColChange (LastRow : Variant, LastCol : Integer) :
*****
*****
dgrdJournalEntries_BeforeRowColChange (Cancel : Integer) :
*****
*****
cmdSearch_Click () :
*****
*****
cmdClear_Click () :
*****
*****
abarContextMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :
*****
*****

```

LOGICAL VIEW REPORT

enableText () :

=**METHOD :** enableText**PURPOSE :** This will enable all the fields**PARMS :****RETURN :**

=**sndPanelMsg (amsgType : panelMsg) :**

=**METHOD :** sndPanelMsg**PURPOSE :** This will display the appropriate message to the panel**PARMS :**

amsgType (panelMsg) = the type of message that should be displayed

RETURN : None

=**bldBusinessTrans () :**

=**METHOD :** bldBusinessTrans**PURPOSE :** This will build the translation tables within the grid**PARMS :****RETURN :** None

=**getAllBusinessData () :**

=**METHOD :** getAllBusinessData**PURPOSE :** this method will get all the businesses from the data base**PARMS :****RETURN :** None

=**Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :**
deleteOffice () :

=**METHOD :** deleteOffice**PURPOSE :** This will delete a record from the data base**PARMS :****RETURN :**

=**saveOffice () :**

=**METHOD :** saveOffice**PURPOSE :** This will determine whether the record will be inserted or updated.A request will be made to the business services

LOGICAL VIEW REPORT

PARMS :**RETURN :**

=

checkForMissingData () : Boolean

=

METHOD : checkForcheckForMissingData**PURPOSE :** This will determine if any required data is missing.**PARMS :****RETURN :** lbBad As Boolean

=

setTextFields () :

=

METHOD : setTextFields**PURPOSE :** This will bind the data**PARMS :****RETURN :**

=

Form_Load () :

=

METHOD : Form_Load**PURPOSE :** This will retrieve all required data, then bind**PARMS :****RETURN :**

=

dgrdOffice_RowColChange (LastRow : Variant, LastCol : Integer) :

=

METHOD : dgrdOffice_RowColChange**PURPOSE :** This will post the event based on whether a record was updated**PARMS :****RETURN :**

=

dgrdOffice_PostEvent (MsgId : Integer) :

=

METHOD : dgrdOffice_PostEvent**PURPOSE :** This message will triggered from the RowColChange. If data was changed the event will be posted. It is here will the grid will be re-built.**PARMS :****RETURN :**

=

LOGICAL VIEW REPORT

Public Operations:

deleteOffice () :
 saveOffice () :
 missingData () : Boolean
 setTextFields () :
 getAllOfficeData () :

Private Operations:

Form_Unload (Cancel : Integer) :
 Form_Load () :
 dgrdOffice_RowColChange (LastRow : Variant, LastCol : Integer) :
 dgrdOffice_PostEvent (MsgId : Integer) :
 dgrdOffice_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
 dgrdOffice_BeforeRowColChange (Cancel : Integer) :
 abarOffice_Click (Tool : ActiveBarLibraryCtl.Tool) :

Form Organization Maint

Private Attributes:

cFORM_MIN_HEIGHT : = 5325
 cFORM_MIN_WIDTH : = 10050
 gbUpdateCorp : Boolean
 gbAddNewOffice : Boolean
 gbAddNewCorp : Boolean
 gstrSearchCriteria : String
 gstrRowName : String

Public Operations:

deleteData () :
 =
 METHOD : deleteData
 PURPOSE : Called from the MDI form, this will call this forms delete method
 PARMS :

RETURN :

newData () :

=
 METHOD : newData
 PURPOSE : Called from the MDI form this will determine if data can be saved
 prior to creating a new record
 PARMS :

RETURN :

LOGICAL VIEW REPORT

PURPOSE : This will cancel an add t the corporg Record set
PARMS :

RETURN :

=

deleteCorpOrg () :

=

METHOD : deleteCorpOrg
PURPOSE : This will delete a record from the Corp_Org Table
PARMS :

RETURN :

=

bldCalendarTranslation () :

=

METHOD : bldCalendarTranslation
PURPOSE : This will build the calendar translation table
PARMS :

RETURN :

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
saveOfficeCorprs () :

PARMS : None

RETURN : None

=

saveOfficeCorp () :

=

METHOD : saveOfficeCorp
PURPOSE : This will determine if the data should inserted or updated.
The appropriate call to the business service will be made
PARMS :

RETURN :

=

checkForMissingData () : Boolean

=

METHOD : checkForMissingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

=

LOGICAL VIEW REPORT

METHOD : dgrdCorpOrg_RowColChange
PURPOSE : If data changed the method will post an event
PARMS :

RETURN :

=
dgrdCorpOrg_PostEvent (MsgId : Integer) :

=
METHOD : dgrdCorpOrg_PostEvent
PURPOSE : This event is posted in the RowColChange event. If data was changed
this event will re-populate the grid.
PARMS :

RETURN :

=
dgrdCorpOrg_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=
METHOD : dgrdCorpOrg_MouseDown
PURPOSE : Determine if the right mouse button was selected. If so will then set
the options appropriately.
PARMS :

RETURN :

=
dgrdCorpOrg_BeforeRowColChange (Cancel : Integer) :

=
METHOD : dgrdCorpOrg_BeforeRowColChange
PURPOSE : Determine if any data missing if none the data will be saved. The
appropriate message will be sent if any data missing.
PARMS :

RETURN :

=
dcboOffice_Click (Area : Integer) :

=
METHOD : dcboOffice_Click
PURPOSE : This will update the Recordset that this drop down is attached to
PARMS :

RETURN :

=
dcboCalendar_Click (Area : Integer) :

=
METHOD : dcboCalendar_Click
PURPOSE : This will update the Recordset that this drop down is attached to
PARMS :

LOGICAL VIEW REPORT

```

=
newParm 0 :
sndPanelMsg (amsgType : panelMsg) :
=
METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
    amsgType (panelMsg) = the type of message that should be displayed
RETURN : None

```

```

=
checkForMissingData 0 : Boolean

```

```

=
METHOD : checkForMissingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

```

```

=
setTextFields 0 :

```

```

=
METHOD : setTextFields
PURPOSE : This method will bind the text fields and drop downs to the data
PARMS :

RETURN :

```

```

=
disableText 0 :
enableText 0 :

```

```

=
METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

RETURN :

```

```

=
bldParmTranslation 0 :

```

```

=
METHOD : bldParmTranslation
PURPOSE : This will build the type translation table
PARMS :

RETURN :

```

```

=
getAllParms 0 :

```

LOGICAL VIEW REPORT

Private Operations:

Form_Load () :

dgrdQEParms_RowColChange (LastRow : Variant, LastCol : Integer) :

frmRuleMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3540

cFORM_MIN_WIDTH : = 10575

Public Operations:

cancelAdd () :

deleteData () :

newData () :

Private Operations:

GetSearchFields () :

ClearSearchFields () :

ManageSearchButton () :

EnableEditFields (bEnable : Boolean) :

processRS_Rules (aAction : frmRuleMaintActions, avParms() : Variant) : frmRuleMaintRC

cmdSearch_Click () :

cmdClear_Click () :

txtRuleName_Change () :

txtRuleDescrip_Change () :

Form_Unload (Cancel : Integer) :

Form_Load () :

dgrdRules_RowColChange (LastRow : Variant, LastCol : Integer) :

dgrdRules_PostEvent (MsgId : Integer) :

dgrdRules_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

dgrdRules_BeforeRowColChange (Cancel : Integer) :

abarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmRuleMaintLines

Private Attributes:

cINVALIDCALLTORULELINES : Variant = cHIGHESTERROR + 1000

cHIGHESTERROR : Variant = vbObjectError + 256

cFORM_MIN_HEIGHT : = 5865

cFORM_MIN_WIDTH : = 9735

mvarRuleAETID : Long

Form Global variables and constants

giNextSeq : Integer

LOGICAL VIEW REPORT

```

=
cmdFetch_Click () :
cmdClear_Click () :

```

```

=
Command Button Code

```

```

=
Form_Load () :

```

```

=
Form_Load

```

frmSLChartofAccounts

```

=
MODULE : frmSLChartofAccounts
PURPOSE : This form will add, update and delete subledger chart of accounts

```

Private Attributes:

```

cFORM_MIN_HEIGHT : = 7020
cFORM_MIN_WIDTH : = 10650
gstrSearchCriteria : String
gstrRowName : String

```

Public Operations:

```

deleteData () :
=
METHOD : deleteData
PURPOSE : This method is called by the MDI form from the tool bar
PARMS :

RETURN : None

```

```

=
newData () :
PARMS :

RETURN : None

```

Private Operations:

```

disableText () :
=
METHOD : disableText

```

LOGICAL VIEW REPORT

PURPOSE : This will disabl all controls
PARMS :

RETURN : None

cancelUpdate () :

METHOD : cancelUpdate
PURPOSE : This will cancel an update
PARMS :

RETURN : None

anyChange () : Boolean
PARMS :

RETURN : Boolean

checkAcctRoll () :

METHOD : checkAcctRoll
PURPOSE : Need to check to see if the account rollup grid changed
PARMS :

RETURN : None

checkTransfer () :

METHOD : checkTransfer
PURPOSE : This method will check to see if the check transfer grid changed
PARMS :

RETURN : None

RecordsetChanges (arsToBeChecked : ADOR.Recordset) : Boolean

METHOD : RecordsetChanges
PURPOSE : This method will check to see if the recordset has changed
PARMS :

arsToBeChecked (ADOR.Recordset) =
RETURN : Boolean

bldAfterSave () :

METHOD : bldAfterSave
PURPOSE : This will re-build the grid after saving the data

LOGICAL VIEW REPORT

bldTransferAndRoll () :

=

METHOD : bldTransferAndRoll**PURPOSE :** This will get all the transfer and roll accounts then build the grid**PARMS :****RETURN :** None

=

bldTdgItemList () :

=

METHOD : bldTdgItemList**PURPOSE :** This will build the Item List**PARMS :****RETURN :** None

=

deleteSubLedger () :

=

METHOD : deleteSubLedger**PURPOSE :** This will delete a subledger**PARMS :****RETURN :** None

=

enableText () :

=

METHOD : enableText**PURPOSE :** This will enable all the controls**PARMS :****RETURN :** None

=

checkForMissingData () : Variant

=

METHOD : checkForMissingData**PURPOSE :** This will determine if there is any missing data**PARMS :****RETURN :** Variant

=

newSubledger () :

=

METHOD : newSubledger**PURPOSE :** This will create a new subledger**PARMS :**

LOGICAL VIEW REPORT

RETURN : None

=

txtSubLedgerName_KeyUp (KeyCode : Integer, Shift : Integer) :

=

METHOD : txtSubLedgerName_KeyUp

PURPOSE : This will check to see the values that were entered in this control

PARMS :

KeyCode [Integer] =

Shift [Integer] =

RETURN : None

=

txtSubLedgerCode_KeyUp (KeyCode : Integer, Shift : Integer) :

=

METHOD : txtSubLedgerCode_KeyUp

PURPOSE :

PARMS :

KeyCode [Integer] =

Shift [Integer] =

RETURN : None

=

tdgItemList_RowColChange (LastRow : Variant, LastCol : Integer) :

=

METHOD : tdgItemList_RowColChange

PURPOSE :

PARMS :

LastRow [Variant] =

LastCol [Integer] =

RETURN : None

=

tdgItemList_PostEvent (MsgId : Integer) :

=

METHOD : tdgItemList_PostEvent

PURPOSE :

PARMS :

MsgId [Integer] =

RETURN : None

=

tdgItemList_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

Method : tdgItemList_MouseUp

Purpose: displays the popup, if there are edit checks will not save pointer

Parms : None

Return : None

if there are edit checks what do we want to d

tdgItemList_BeforeRowColChange (Cancel : Integer) :

=
METHOD : tdbgItemList_BeforeRowColChange
PURPOSE :
PARMS :
 Cancel [Integer] =
RETURN : None

tdbgTransfer_Click () :

=
METHOD : tdbgTransfer_Click
PURPOSE : This will re-position the grid if the value is empty
PARMS :

RETURN : None

tdbgTransfer_AfterColEdit (ColIndex : Integer) :

=
METHOD : tdbgTransfer_AfterColEdit
PURPOSE :
PARMS :
 ColIndex [Integer] =
RETURN : None

tdbgAcctRollUp_Click () :

PARMS :

RETURN : None

tdbgAcctRollUp_AfterColEdit (ColIndex : Integer) :

=
METHOD : tdbgAcctRollUp_AfterColEdit
PURPOSE :
PARMS :
 ColIndex [Integer] =
RETURN : None

Form_Unload (Cancel : Integer) :

=
METHOD : Form_Unload
PURPOSE :
PARMS :
 Cancel [Integer] =
RETURN : None

LOGICAL VIEW REPORT

frmSLChartGroups

=

MODULE : frmSLChartGroups

PURPOSE : This form will allow you to add,update,delete and display subledger groups.
Subledgergroups contain subledger chart of accounts

=

Private Attributes:

cFORM_MIN_HEIGHT : = 8160

cFORM_MIN_WIDTH : = 10620

gbUpdateGroup : Boolean

gbAddNewSLChart : Boolean

gbAddNewGroup : Boolean

gstrSearchCriteria : String

gstrRowName : String

Public Operations:

chkFerMissingSLChart () : Boolean

deleteData () :

=

METHOD : deleteData

PURPOSE : The purpose of this method is to delete data

PARMS :

RETURN : None

=

newData () :

=

METHOD : newData

PURPOSE : This is called by the MDI tool bar

PARMS :

RETURN : None

=

Private Operations:

disableText () :

=

METHOD : disableText

PURPOSE : This method will disable the controls on the window

PARMS :

RETURN : None

=

newSLChartForGroup () :

PARMS : None

LOGICAL VIEW REPORT

saveGroupRs 0 :

PARMS : None

RETURN : None

checkForMissingData 0 : Boolean

PARMS : None

RETURN : Boolean

bindSLGroupFlds 0 :

METHOD : bindSLGroupFlds

PURPOSE : This method will bind the group text fields to the record set

PARMS : None

RETURN : None

newSLGroup 0 :

PARMS : None

RETURN : None

enableText 0 :

METHOD : enableText

PURPOSE : This method will enable the input text fields

PARMS : None

RETURN : None

setTextFields 0 :

METHOD : setTextFields

PURPOSE : This method will bind the text fields to the record sets

PARMS : None

RETURN : None

txtSLSearchGroup_KeyUp (KeyCode : Integer, Shift : Integer) :

METHOD : txtSLSearchGroup_KeyUp

PURPOSE : This method will determine if the input entered length is greater than 2

PARMS :

KeyCode [Integer] =

Shift [Integer] =

RETURN : None

-C48-

LOGICAL VIEW REPORT

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
getSLChartOfAccounts (aSearch : groupID) :

PARMS : None

aSearch [groupID] = if a groupid was supplied

RETURN : None

getSLGroupData () :

METHOD : getSLGroupData

PURPOSE : This method will retrieve the subledger Group data

PARMS : None

RETURN : None

bldSLCTranslation () :

METHOD : bldSLCTranslation

PURPOSE : This method will build the translation for the SLChartAccounts Grid

PARMS : None

RETURN : None

Form_Load () :

PARMS : None

RETURN : None

dgrdSLGroup_RowColChange (LastRow : Variant, LastCol : Integer) :

PARMS :

LastRow [Variant] =

LastCol [Integer] =

RETURN : None

dgrdSLGroup_PostEvent (MsgId : Integer) :

RETURN : None

dgrdSLGroup_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

METHOD : dgrdSLGroup_MouseUp

PURPOSE : This method will determine what options are available on the context menu

PARMS : None

Button [Integer] =

Shift [Integer] =

X [Single] =

Y [Single] =

RETURN : None

LOGICAL VIEW REPORT

modMain

Public Attributes:

```

HH_TP_HELP_WM_HELP : = &H11
HH_TP_HELP_CONTEXTMENU : = &H10
HH_HELP_CONTEXT : = &HF
HH_DISPLAY_TEXT_POPUP : = &HE
HH_GET_WIN_HANDLE : = &H6
HH_GET_WIN_TYPE : = &H5
HH_SET_WIN_TYPE : = &H4
HH_DISPLAY_TOPIC : = &H0
HELP_MAP_AE_WELCOME : Integer = 1
cBAD_ENTRY_BACKCOLOR : = vbYellow
cDISABLED_BACKCOLOR : = &H8000000F
cENABLED_BACKCOLOR : = &H80000005

```

Color Constants

```

UNLEN : = 256
GWW_HWNDPARENT : = (-8)

```

Public Operations:

```

activeBarLoad () :
disableActiveBar () :
RecordsetChanged (arsToBeChecked : ADOR.Recordset) : Boolean
pGetUserName () : String
htmlhelp (hwndCaller : Long, pszFile : String, uCommand : Long, dwData : Long) : Long
    This Declare used for help window
SetWindowWord (hwnd : Long, nIndex : Long, wParam : Long) : Long
    This declare used for floatable window (frmerrors)
GetUserName (lpBuffer : String, nSize : Long) : Long

```

	byval alJEID -
	alProductID -
	alBankID -
	acurTXNAmount -
	astrDRCPIND -
	alCOAID -
	astrPostPeriod -
<u>Outputs:</u>	None
<u>Returns:</u>	None

CreateSLMonthsBals(long byval SLBalanceID, currency byval acurTXNAmount, string byval astrPostPeriod)

<u>Class:</u>	IPostSL
<u>Description:</u>	None
<u>Inputs:</u>	byval SLBalanceID -
	byval acurTXNAmount -
	byval astrPostPeriod -
<u>Outputs:</u>	None
<u>Returns:</u>	None

Finalize Processing()

<u>Class:</u>	IPostSL
<u>Description:</u>	None
<u>Inputs:</u>	None
<u>Outputs:</u>	None
<u>Returns:</u>	None

WHAT IS CLAIMED IS:

1. A method of asset level accounting using a lease and loan sub-ledger accounting system (10), the accounting system including a lease and loan accounting engine (12), a plurality of sub-ledger accounting components independent from the accounting engine, and a plurality of programmatic interfaces (140) enabling
5 communication with components of the accounting engine, said accounting system running within an operational system, said method comprising the steps of:

isolating accounting functions from the operational system; and

providing sub-ledger transaction detail.

2. A method according to Claim 1 further comprising the step of
10 providing multi-national detail.

3. A method according to Claim 1 further comprising the step of internally and externally referring to financial entities.

4. A method according to Claim 1 further comprising the step of supporting multiple pricing models.

5. A method according to Claim 1 further comprising the step of defining
15 and adding information needed to support specific accounting requirements.

6. A method according to Claim 1 further comprising the step of identifying every transaction in the accounting system (10) using an audit transaction component (62).

7. A method according to Claim 6 further comprising the step of relating
20 every accounting transaction with a corresponding operational transaction using an operational system (60) enabled with an audit transaction component (62).

a plurality of sub-ledger accounting components independent from said accounting engine; and

a plurality of programmatic interfaces enabling communication of said sub- ledger accounting components with said accounting engine.

5 18. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components provides multi-national detail.

 19. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises internal and external references to financial entities.

10 20. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system supports multiple pricing models and multiple operational systems.

 21. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises capability
15 for a user to define and add information needed to support specific accounting requirements.

 22. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises an audit transaction component (62) identifying every transaction in said accounting system.

20 23. A system (10) according to Claim 22 wherein said audit transaction component (62) allows an operational system to relate every accounting transaction with a corresponding operational transaction.

 24. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises a flexible
25 event driven process model (50) to allow an accounting system to derive a correct accounting entry for a lease or loan accounting event.

33. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises stream representations (100) of compressed data.

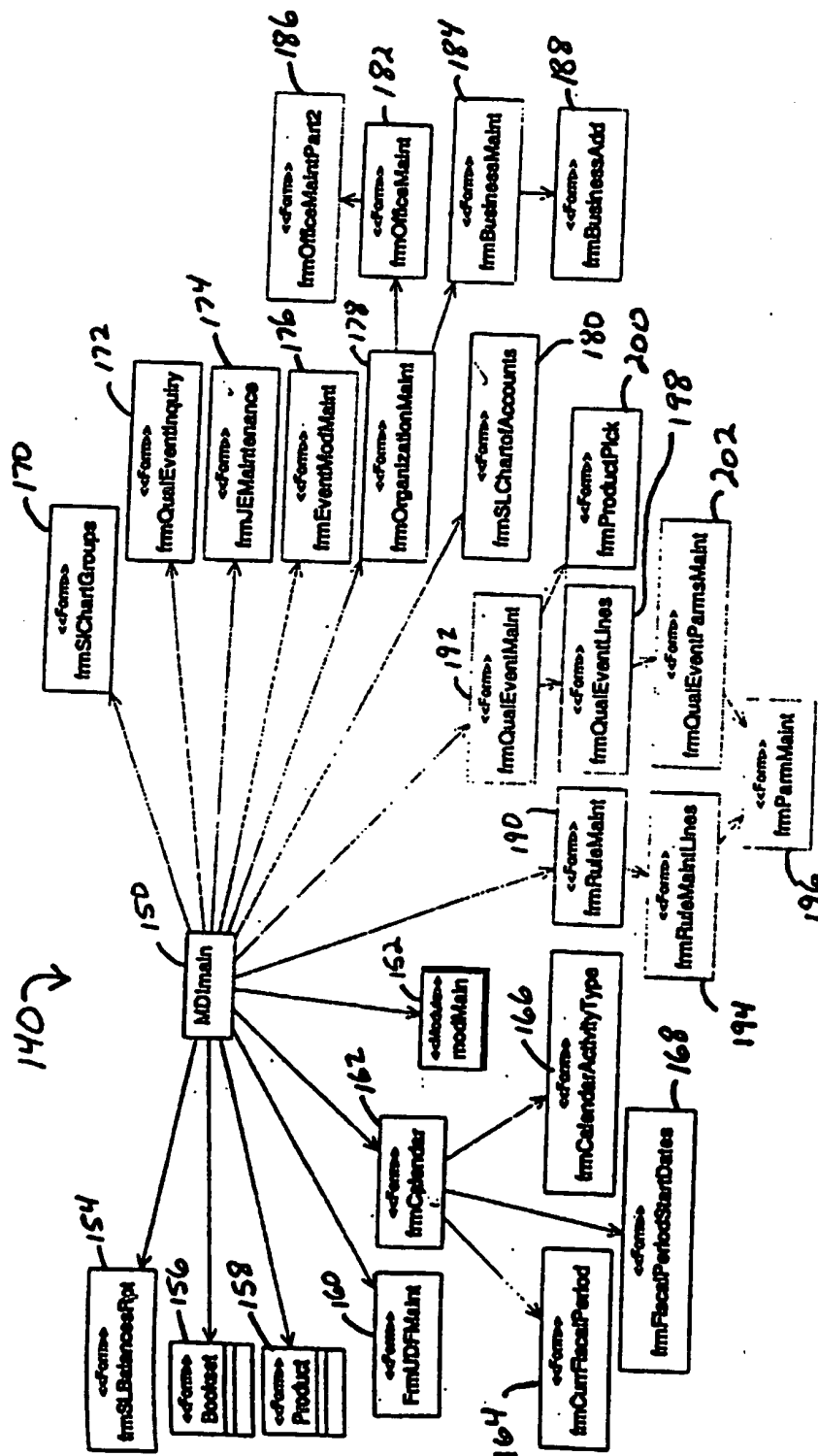


FIG. 2